AutoCFR: Learning to Design Counterfactual Regret Minimization Algorithms

Hang Xu[1,2*], Kai Li[1,2*], Haobo Fu[4], Qiang Fu[4], Junliang Xing[1,2,3†]

[1]Institute of Automation, Chinese Academy of Sciences
[2]School of Artificial Intelligence, University of Chinese Academy of Sciences
[3]Tsinghua University [4]Tencent AI Lab
{xuhang2020, kai.li}@ia.ac.cn, {haobofu, leonfu}@tencent.com, jlxing@tsinghua.edu.cn

Abstract

Counterfactual regret minimization (CFR) is the most commonly used algorithm to approximately solving two-player zero-sum imperfect-information games (IIGs). In recent years, a series of novel CFR variants such as CFR+, Linear CFR, DCFR have been proposed and have significantly improved the convergence rate of the vanilla CFR. However, most of these new variants are hand-designed by researchers through trial and error based on different motivations, which generally requires a tremendous amount of efforts and insights. This work proposes to *meta-learn* novel CFR algorithms through evolution to ease the burden of manual algorithm design. We first design a search language that is rich enough to represent many existing hand-designed CFR variants. We then exploit a scalable regularized evolution algorithm with a bag 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 IIGs not seen during training and performs on par with or better than existing state-of-the-art CFR variants. The code is available at https://github.com/rpSebastian/AutoCFR.

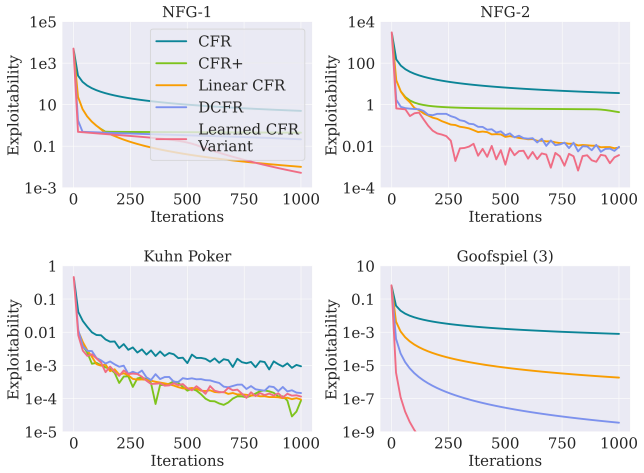

Figure 1: Convergence speed of four CFR-type algorithms and our learned one in four games. Different CFR-type algorithms perform distinctively in these games. Our framework learns a new CFR variant performing consistently well.

Introduction

Games in extensive form provide a mathematical framework for modeling sequential decision-making problems with imperfect information. Solving such imperfect-information games (IIGs) requires reasoning under uncertainty about the opponents' hidden information. The hidden information is omnipresent in real-world strategic interactions, such as negotiation, business, and security, making the research on IIGs crucial both theoretically and practically.

In this work, we focus on solving two-player zero-sum IIGs. For these games, the common goal is to find an (approximate) Nash equilibrium (NE) (Nash 1950) in which no player can improve by deviating from this equilibrium. As a popular method of computing NE, counterfactual regret minimization (CFR) (Zinkevich et al. 2007) 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 gradually approximates the NE. Over the past decade, many novel CFR variants have been proposed (Tammelin 2014; Brown and Sandholm 2019; Li et al. 2020; Farina et al. 2019) with faster convergence than the vanilla CFR. For example, CFR+ (Tammelin 2014; Bowling et al. 2015) was the key to solve the heads-up limit Texas Hold'em poker. Discount CFR (DCFR) (Brown and Sandholm 2019) is a family of algorithms that assigns more weights to the later iterations and obtains competitive performance compared with other CFR variants. Linear CFR (Brown and Sandholm 2019; Brown et al. 2019) is a simplified version of DCFR and performs well in practice.

Despite the great success of CFR and its existing variants, most 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.*, regrets 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 IIGs. As shown in Figure 1, it is clear that CFR variants perform differently in various IIGs, and no one performs consistently well in

*Equal contribution. † Corresponding author.

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.

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. More specifically, we formulate 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 IIGs. The objective of the outer loop is to minimize the distance between the strategy obtained by the inner loop and the NE in each meta-training IIG. Our ultimate goal is to discover novel CFR variants that can generalize across many different IIGs, instead of specific to the particular training domain. To this end, we design a search language that can express general symbolic functions applied to any IIGs. This language is expressive enough to represent many previously proposed hand-designed CFR variants (*e.g.*, CFR+, DCFR, Linear CFR, *etc.*). Since efficiently searching over the space of algorithms defined by this language is generally difficult, we exploit the scalable regularized evolution (Real et al. 2019) 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 (Real et al. 2020; Co-Reyes et al. 2020). We adapt this method to design algorithms for equilibrium finding in IIGs 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 novel contributions:

- We propose AutoCFR, the first framework to meta-learn novel CFR-type algorithms.
- We design an expressive language to describe the space of CFR-type algorithms and exploit an efficient search algorithm to make the search feasible.
- We automatically discover a new CFR variant with strong generalization ability, achieving competitive performance on new IIGs not seen during training.

Preliminary

Here we present some background knowledge needed for the rest of the paper. In this section, we first provide some notations to formulate IIGs. Next, we introduce some important concepts like best response, Nash equilibrium, and exploitability. Finally, we discuss the vanilla CFR algorithm and its typical variants.

Notations

IIGs are usually described by a tree-based formalism called extensive-form games (Osborne and Rubinstein 1994). 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 IIGs, 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.*, $\pi^\sigma(h) = \pi^\sigma_i(h)\pi^\sigma_{-i}(h)$, where $\pi^\sigma_i(h) = \prod_{h'a\sqsubseteq h, \mathcal{P}(h')=i} \sigma_i(h', a)$ is player i's contribution and $\pi^\sigma_{-i}(h) = \prod_{h'a\sqsubseteq h, \mathcal{P}(h')\neq i} \sigma_{\mathcal{P}(h')}(h', a)$ 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 history h' to h is defined as $\pi^\sigma(h', h) = \pi^\sigma(h)/\pi^\sigma(h')$ if $h' \sqsubseteq h$. $\pi^\sigma_i(I_i), \pi^\sigma_{-i}(I_i), \pi^\sigma_i(h, h'), \pi^\sigma_{-i}(h, h')$ are defined similarly.

Best Response and Nash Equilibrium

The **best response** to σ_{-i} is any strategy $\text{BR}(\sigma_{-i})$ 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 $\sigma^* = (\sigma^*_i, \sigma^*_{-i})$ 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 $e_i(\sigma_i) = u_i(\sigma^*_i, \sigma^*_{-i}) - u_i(\sigma_i, \text{BR}(\sigma_i))$. In an **$\epsilon$-Nash equilibrium**, no player has exploitability higher than ϵ. The exploitability of a strategy profile σ is $e(\sigma) = \sum_{i\in\mathcal{N}} e_i(\sigma_i)/|\mathcal{N}|$. It can be interpreted as the approximation error to the Nash equilibrium.

Counterfactual Regret Minimization

Counterfactual Regret Minimization (CFR) is an iterative algorithm for computing approximate Nash equilibrium

in extensive-form IIGs (Zinkevich et al. 2007). 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 (Hart and Mas-Colell 2000; Cesa-Bianchi and Lugosi 2006) to compute the new strategy for the next iteration:

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

where $R_i^{T,+}(I,a) = \max(R_i^T(I,a), 0)$. In two-player zero-sum IIGs, 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 (Zinkevich et al. 2007). $\bar{\sigma}^T$ is calculated as:

$$C_i^T(I,a) = \sum_{t=1}^T \left(\pi_i^{\sigma^t}(I)\sigma_i^t(I,a)\right), \bar{\sigma}_i^T(I,a) = \frac{C_i^T(I,a)}{\sum_{a' \in \mathcal{A}(I)} C_i^T(I,a')}, \tag{2}$$

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

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+ (Tammelin 2014; Bowling et al. 2015) is like CFR with two 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. DCFR (Brown and Sandholm 2019) 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 (Brown and Sandholm 2019) is a special case of DCFR where iteration T's contribution to cumulative regrets and cumulative strategy is proportional to T. ECFR (Li et al. 2020) 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 $w(I,a) = \exp(r_i(I,a) - 1/|\mathcal{A}(I)| \sum_{a \in \mathcal{A}(I)} r_i(I,a))$.

Learning Novel CFR Variants

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.

The AutoCFR Framework

As mentioned earlier, CFR and its variants have obtained remarkable performance in solving IIGs. This success was possible due to decades of persistent efforts by researchers in the game theory and machine learning communities. However, 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 consideration, 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 IIGs $\mathbb{G} = \{G_i\}_{i=1}^N$, 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 IIGs $\hat{\mathbb{G}}$. Formally, AutoCFR's training objective function is:

$$A^* = \arg\max_{A \in \mathbb{A}} \left[\sum_{G \in \mathbb{G}} W_G \operatorname{Eval}(A, G) \right], \tag{3}$$

where $\operatorname{Eval}(A,G)$ is the inner loop procedure that evaluates algorithm A's performance in the training IIG G. W_G represents the weight of G, which is proportional to the game size. To calculate $\operatorname{Eval}(A,G)$, 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.*,

$$\operatorname{Eval}(A,G) = \min\left(S_G, \frac{\log E_{\text{CFR}}^G - \log E_A^G}{\log E_{\text{CFR}}^G - \log E_{\text{DCFR}}^G}\right), \tag{4}$$

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, proportional to the game size. We exploit the logarithmic function because CFR converges asymptotically at a logarithmic rate. The minimum function is used to avoid A overfitting to G.

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 IIGs $\mathbb{G}$. 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 such a rich space of algorithms and diverse training IIGs, we will automatically discover novel, efficient, and generalizable CFR variants.

Search Language

Each iteration T of the CFR-type algorithms consists of two steps, *i.e.*, policy evaluation and policy improvement. 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. 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 IIGs. Similar to (Alet et al. 2019; Co-Reyes et al. 2020), 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 `PI` (policy improvement). 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.
- **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.

Figure 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.

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 (Eqn. 3) non-smooth, and therefore the space hard to search. We use the regularized evolution algorithm (Real et al. 2019) as the search method due to its simplicity and efficiency for this type of search problems, which has made remarkable breakthroughs (Piergiovanni et al. 2019; Faust, Francis, and Mehta 2019; Co-Reyes et al. 2020; Franke et al. 2020; Ci et al. 2021) 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.

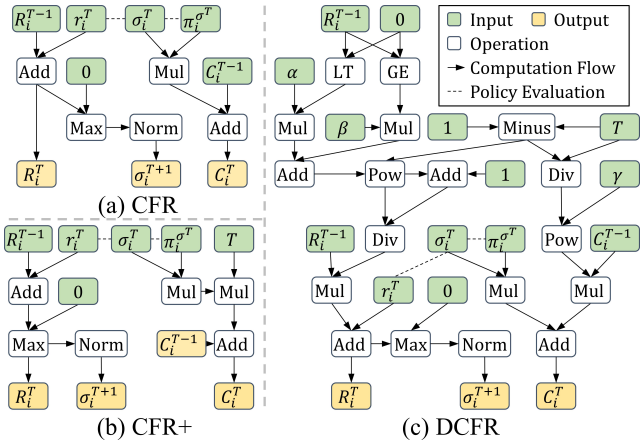

Figure 2: Our designed search language can represent existing CFR variants. (a)(b)(c) visualizes the computational graphs of CFR, CFR+, and DCFR. Moreover, this language enables the description of potentially better CFR variants.

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 $\mathrm{Eval}(A, G)$, 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 (Hutter, Kotthoff, and Vanschoren 2019), 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

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**
  $\hat{A}.score \leftarrow \sum_{G \in \mathbb{G}} W_G \, \mathrm{Eval}(\hat{A}, G)$ ▷(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 $A' \leftarrow \mathrm{Mutate}(A)$;
  $A'.valid \leftarrow \mathrm{ValidityCheck}(A')$;
  $A'.hurdle\_score \leftarrow \mathrm{Eval}(A', G_h)$;
  $A'.hash \leftarrow \mathrm{HashEncoding}(A')$;
  hurdle threshold $\eta \leftarrow \mathrm{Percentile}(\mathbb{P}, 75^{th})$;
  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
      $A'.score \leftarrow \sum_{G \in \mathbb{G}} W_G \, \mathrm{Eval}(A', G)$;
    Add A' to $\mathbb{H}$ and the circular queue $\mathbb{P}$;

Output: A^* with the highest score

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', 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 IIGs, 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 $\mathrm{Eval}(A, G_h) < \eta$, 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.

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

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**
  $\pi^{\sigma^T}, r^T \leftarrow A.\texttt{PE}(G, \sigma^T)$;
  $inputs = \{\sigma^T, R^{T-1}, \pi^{\sigma^T}, C^{T-1}, r^T, T, \ldots\}$;
  $\sigma^{T+1}, R^T, C^T \leftarrow A.\texttt{PI}(inputs)$;
$\bar{\sigma}^M \leftarrow \mathrm{Normalize}(C^M)$;
$E^G_A \leftarrow \bar{\sigma}^M$'s exploitability on G;

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

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.

Experiments

We first describe the experimental setup, including training games, testing games, and training details. We then analyze the characteristics of the learned algorithm and compare it with state-of-the-art CFR variants. Finally, we conduct some ablations to understand the settings of our framework.

Experimental Setup

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 training games should not be too large to solve as AutoCFR will evaluate thousands of CFR-type algorithms during training.

We use some commonly used extensive-form games in the IIG research community. ***Kuhn Poker*** is a simplified form of poker, with three cards in a deck and one chance to bet for each player. ***Leduc Poker*** is a larger game with a 6-card deck and two rounds. In ***Liar's Dice (x)***, each player gets an x-sided dice, rolls them at the start, and then takes turns placing bets on the outcome. ***Goofspiel (x)*** is a card game where each player has x cards and tries to obtain more points by making sealed bids in x rounds. ***HUNL Subgame (x)*** [1] is a heads-up no-limit Texas hold'em (HUNL) sub-game generated by Libratus (Brown and Sandholm 2017, 2018).

In addition, we manually design some normal-form games (***NFG-{1-4}***). For example, *NFG-1* is a simple two-action game where players decide between two actions.

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

Player 1's payoff is 2 if takes the first action, and is 20,000 or 1 if takes the second action, depending on player 2. It 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. 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*.

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*, *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.*, *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.

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 (Co-Reyes et al. 2020). The parent program mutates with 0.95 probability and remains the same otherwise. 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 $\mathrm{Eval}(A, G)$, 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.

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:

$$
\begin{aligned}
R_i^T(I,a) &= \max\left(0, R_i^{T-1}(I,a) * \frac{(T-1)^{1.5}}{(T-1)^{1.5}+1.5} + r_i^T(I,a)\right), \\
C_i^T(I,a) &= C_i^{T-1}(I,a) * \frac{T-1}{T} + \pi_i^{\sigma^T} * T^3 * \sigma_i^T(I,a), \\
\sigma_i^{T+1}(I,a) &= \frac{R_i^{T,+}(I,a)}{\sum_{a' \in \mathcal{A}(I)} R_i^{T,+}(I,a')}.
\end{aligned}
\tag{5}
$$

DCFR+'s improvement over existing CFR variants shown in Figures 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, 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 actions, *i.e.*, actions that cause huge mistakes. It is worth noting that D-CFR'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 get 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.

	AutoCFR	AutoCFR-4	AutoCFR-S
Kuhn Poker	1.1489e-4	**5.7798e-5**	1.4492e-4
Goofspiel (4)	**1.0431e-6**	2.7402e-6	3.3271e-4
Leduc Poker	**1.4592e-6**	2.8801e-6	1.1364e-2
Subgame (3)	**2.2016e-5**	4.3293e-5	4.8060e-3
Subgame (4)	**9.8044e-6**	4.3926e-5	7.5901e-3

Table 1: Performance comparison of the best algorithms learned by three AutoCFR variants.

Ablation Studies

Varying the number of training games: We consider how the number of training games affects the learned algorithm. We compare learning with eight games versus with four extensive-form games. The results are shown in Table 1. AutoCFR-4, *i.e.*, the learned algorithm using four games, performs best in the training game *Kuhn Poker*; however, it performs relatively poorly than AutoCFR, *i.e.*, the learned algorithm DCFR+ using eight games, in the testing games. It is clear that training with four games suffers from some over-fitting, and the additional four normal-form games increase the learned algorithm's generalization performance.

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. As shown in Table 1, bootstrapping from existing CFR variants significantly improves the learning performance of AutoCFR over AutoCFR-S without bootstrapping. In Figure 5, 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 0.5) even

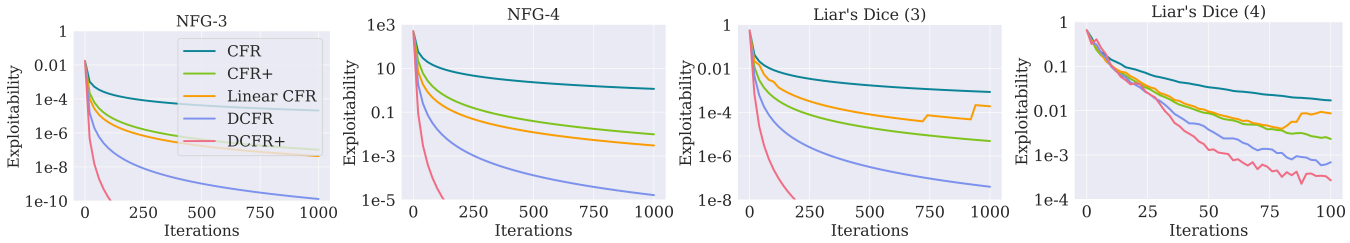

Figure 3: Comparison of DCFR+ against four CFR variants on eight training games. Another four training games are in Fig 1.

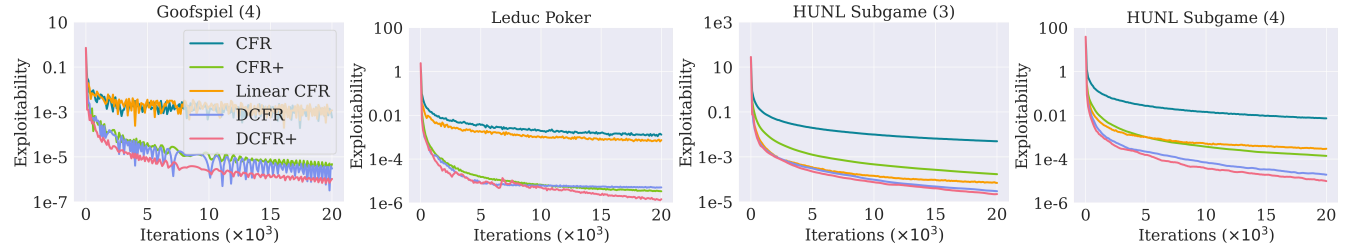

Figure 4: Comparison of DCFR+ against four CFR variants on four testing games.

by learning from scratch. Although there is still much room for improvement in learning from scratch, we believe this is mainly because AutoCFR-S only explores a tiny proportion of the vast search space due to the limited budget available for training machines. We plan to invest more computational resources and improve the framework's search efficiency to discover better and more inspiring CFR variants.

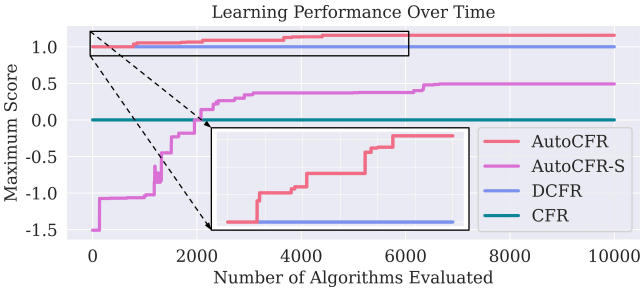

Figure 5: Performance comparison between bootstrapping (AutoCFR) and learning from scratch (AutoCFR-S).

Related Work

AutoML (Yao et al. 2019; He, Zhao, and Chu 2021) aims to automate the machine learning process and has recently become a hot topic in academia and industry. One notable example is neural architecture search (Zoph and Le 2017; Brock et al. 2018; Real et al. 2019) which has achieved great success on computer vision tasks. More recently, AutoML-Zero (Real et al. 2020) automatically finds machine learning algorithms from scratch using basic mathematical operations. AutoCFR shares similar ideas but is applied to search equilibrium-finding algorithms in IIGs for the first time.

Evolutionary algorithm (Goldberg and Deb 1991) is a generic population-based metaheuristic optimization algorithm. It can effectively address complex problems that traditional optimization algorithms struggle to solve. Recent progress using evolutionary algorithms has achieved impressive results in playing games (Kelly and Heywood 2017; Wilson et al. 2018; Silver et al. 2020), optimizing neural networks (Cui et al. 2018; Young et al. 2019), and finding novel reinforcement learning algorithms (Franke et al. 2020; Co-Reyes et al. 2020). In contrast, we use the evolutionary algorithm with lots of acceleration techniques to search for novel CFR-type algorithms for solving IIGs.

Conclusion and Future Work

This work presents an AutoCFR framework for learning to design novel CFR variants. We formalize an expressive language for representing CFR-type algorithms as computational graphs. We then exploit an optimized regularized evolution algorithm to search over the space of computational graphs efficiently. The learned DCFR+ algorithm obtains good generalization performance over a wide range of testing games. Currently, AutoCFR's search space does not include the Monte Carlo CFR type algorithms, and the training and testing games can be expanded to include more diverse types of IIGs. We leave these directions for future research.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China under Grant No. 2020AAA0103401, in part by the Natural Science Foundation of China under Grant No. 62076238 and 61902402, in part by the CCF-Tencent Open Fund, and in part by the Strategic Priority Research Program of Chinese Academy of Sciences under Grant No. XDA27000000.

References

Alet, F.; Schneider, M. F.; Lozano-Perez, T.; and Kaelbling, L. P. 2019. Meta-learning curiosity algorithms. In *International Conference on Learning Representations*, 1–21.

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

Brock, A.; Lim, T.; Ritchie, J. M.; and Weston, N. J. 2018. SmaSH: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*, 1–21.

Brown, N.; Lerer, A.; Gross, S.; and Sandholm, T. 2019. Deep counterfactual regret minimization. In *International Conference on Machine Learning*, 793–802.

Brown, N.; and Sandholm, T. 2017. Safe and nested subgame solving for imperfect-information games. In *Advances in Neural Information Processing Systems*, 689–699.

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

Brown, N.; and Sandholm, T. 2019. Solving imperfect-information games via discounted regret minimization. In *AAAI Conference on Artificial Intelligence*, 1829–1836.

Cesa-Bianchi, N.; and Lugosi, G. 2006. *Prediction, learning, and games*. Cambridge university press.

Ci, Y.; Lin, C.; Sun, M.; Chen, B.; Zhang, H.; and Ouyang, W. 2021. Evolving search space for neural architecture search. In *International Conference on Computer Vision*, 6659–6669.

Co-Reyes, J. D.; Miao, Y.; Peng, D.; Real, E.; Le, Q. V.; Levine, S.; Lee, H.; and Faust, A. 2020. Evolving reinforcement learning algorithms. In *International Conference on Learning Representations*, 1–15.

Cui, X.; Zhang, W.; Tüske, Z.; and Picheny, M. 2018. Evolutionary stochastic gradient descent for optimization of deep neural networks. In *Advances in Neural Information Processing Systems*, 6051–6061.

Farina, G.; Kroer, C.; Brown, N.; and Sandholm, T. 2019. Stable-predictive optimistic counterfactual regret minimization. In *International Conference on Machine Learning*, 1853–1862.

Faust, A.; Francis, A.; and Mehta, D. 2019. Evolving rewards to automate reinforcement learning. arXiv:1905.07628.

Franke, J. K.; Koehler, G.; Biedenkapp, A.; and Hutter, F. 2020. Sample-efficient automated deep reinforcement learning. In *International Conference on Learning Representations*, 1–23.

Goldberg, D. E.; and Deb, K. 1991. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*, 1: 69–93.

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

He, X.; Zhao, K.; and Chu, X. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212: 106622.

Hutter, F.; Kotthoff, L.; and Vanschoren, J. 2019. *Automated machine learning: methods, systems, challenges*. Springer Nature.

Kelly, S.; and Heywood, M. I. 2017. Multi-task learning in Atari video games with emergent tangled program graphs. In *Genetic and Evolutionary Computation Conference*, 195–202.

Li, H.; Wang, X.; Qi, S.; Zhang, J.; Liu, Y.; Wu, Y.; and Jia, F. 2020. Solving imperfect-information games via exponential counterfactual regret minimization. arXiv:2008.02679.

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

Osborne, M. J.; and Rubinstein, A. 1994. *A course in game theory*. MIT press.

Piergiovanni, A.; Angelova, A.; Toshev, A.; and Ryoo, M. S. 2019. Evolving space-time neural architectures for videos. In *International Conference on Computer Vision*, 1793–1802.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2019. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence*, 4780–4789.

Real, E.; Liang, C.; So, D.; and Le, Q. 2020. AutoML-Zero: Evolving machine learning algorithms from scratch. In *International Conference on Machine Learning*, 8007–8019.

Silver, T.; Allen, K. R.; Lew, A. K.; Kaelbling, L. P.; and Tenenbaum, J. 2020. Few-shot Bayesian imitation learning with logical program policies. In *AAAI Conference on Artificial Intelligence*, 10251–10258.

Tammelin, O. 2014. Solving large imperfect information games using CFR+. arXiv:1407.5042.

Wilson, D. G.; Cussat-Blanc, S.; Luga, H.; and Miller, J. F. 2018. Evolving simple programs for playing Atari games. In *Genetic and Evolutionary Computation Conference*, 229–236.

Yao, Q.; Wang, M.; Chen, Y.; Dai, W.; Li, Y.-F.; Tu, W.-W.; Yang, Q.; and Yu, Y. 2019. Taking human out of learning applications: A Survey on automated machine learning. arXiv:1810.13306.

Young, S. R.; Devineni, P.; Parsa, M.; Johnston, J. T.; Kay, B.; Patton, R. M.; Schuman, C. D.; Rose, D. C.; and Potok, T. E. 2019. Evolving energy efficient convolutional neural networks. In *International Conference on Big Data*, 4479–4485.

Zinkevich, M.; Johanson, M.; Bowling, M.; and Piccione, C. 2007. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems*, 1729–1736.

Zoph, B.; and Le, Q. V. 2017. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 1–16.