# Reinforcement Nash Equilibrium Solver

**Xinrun Wang**[1] , **Chang Yang**[2] , **Shuxin Li**[1] , **Pengdeng Li**[1] ,
**Xiao Huang**[2] , **Hau Chan**[3] and **Bo An**[1,4]
[1]Nanyang Technological University, Singapore
[2]The Hong Kong Polytechnic University, Hong Kong SAR, China
[3]University of Nebraska-Lincoln, Lincoln, Nebraska, United States
[4]Skywork AI, Singapore
{xinrun.wang, shuxin.li, pengdeng.li, boan}@ntu.edu.sg, chang.yang@connect.polyu.hk
xiaohuang@comp.polyu.edu.hk, hchan3@unl.edu

## Abstract

Nash Equilibrium (NE) is the canonical solution concept of game theory, which provides an elegant tool to understand the rationalities. Though mixed strategy NE exists in any game with finite players and actions, computing NE in two- or multi-player general-sum games is PPAD-Complete. Various alternative solutions, e.g., Correlated Equilibrium (CE), and learning methods, e.g., fictitious play (FP), are proposed to approximate NE. For convenience, we call these methods as "inexact solvers", or "solvers" for short. However, the alternative solutions differ from NE and the learning methods generally fail to converge to NE. Therefore, in this work, we propose REinforcement Nash Equilibrium Solver (RENES), which *trains a single policy to modify the games with different sizes and applies the solvers on the modified games where the obtained solution is evaluated on the original games*. Specifically, our contributions are threefold. i) We represent the games as $\alpha$-rank response graphs and leverage graph neural network (GNN) to handle the games with different sizes as inputs; ii) We use tensor decomposition, e.g., canonical polyadic (CP), to make the dimension of modifying actions fixed for games with different sizes; iii) We train the modifying strategy for games with the widely-used proximal policy optimization (PPO) and apply the solvers to solve the modified games, where the obtained solution is evaluated on original games. Extensive experiments on large-scale normal-form games show that our method can further improve the approximation of NE of different solvers, i.e., $\alpha$-rank, CE, FP and PRD, and can be generalized to unseen games.

## 1 Introduction

Game theory provides a pervasive framework to model the interactions between multiple players [Fudenberg and Tirole, 1991]. The canonical solution concept in non-cooperative games, i.e., the players try to maximize their own utility, is Nash Equilibrium (NE), where no player can change its strategy unilaterally to increase its own utility [Nash Jr, 1950]. According to Roger Myerson, the introduction of NE is a watershed event for game theory and economics [Myerson, 1999]. NE provides an impetus to understand the rationalities in much more general economic contexts and lies at the foundation of modern economic thoughts [Myerson, 1999; Goldberg *et al.*, 2013]. Mixed strategy NE exists in any game with finite players and actions [Nash Jr, 1950]. However, from an algorithmic perspective, computing NE in two-player or multi-player general-sum games is PPAD-Complete [Daskalakis *et al.*, 2009; Chen *et al.*, 2009]. In two-player zero-sum games, NE can be computed in polynomial time via linear programming. In more generalized cases, the Lemke–Howson algorithm is the most recognized combinatorial method [Lemke and Howson, 1964], while using this algorithm to identify any of its potential solutions is PSPACE-complete [Goldberg *et al.*, 2013].

Given the difficulties of computing NE directly, there are many alternative solutions and learning methods proposed to approximate NE. For convenience, we call these solutions and learning methods as "inexact solvers", or "solvers" for short. Among them, one of the most widely used solution concepts is Correlated Equilibrium (CE) [Aumann, 1974; Aumann, 1987], which considers the Bayesian rationality and is a more general solution concept than NE, i.e., any NE is CE. Recently, another solution concept $\alpha$-rank is proposed [Omidshafiei *et al.*, 2019], which adopts Markov-Conley Chains to highlight the presence of cycles in game dynamics and attempts to compute stationary distributions as a mean for strategy profile ranking. $\alpha$-rank is successfully applied to policy space response oracle (PSRO) to approximate NE [Muller *et al.*, 2020]. The computations of both solution concepts are significantly simplified, but both solutions may be diverge from the accurate NE.

To address the computational challenges, various heuristic methods are proposed to approximate NE. One of the most widely used methods is fictitious play [Brown, 1951; Heinrich *et al.*, 2015], which takes the average of the best-responses iteratively computed by assuming the opponents play the empirical frequency policies. Another widely used learning method is projected replicator dynamics (PRD) [Lanctot *et al.*, 2017], which is an advanced variant of replicator dynamics (RD) [Taylor and Jonker, 1978] by enforcing explo-

rations. Unfortunately, both methods fail to converge to NE in general cases. Another line of research is the homotopy method [Herings and Peeters, 2010; Gemp *et al.*, 2022], which tries to build a continuum between the game and a simplified game with known NE and then uses the known NE to approximate the NE in the original game. The methods to build the continuum is important, which requires game-specific knowledge for efficient approximation. Furthermore, most previous methods require running the methods on each specific game and the obtained models are game-specific and lack the generalizability. Due to the limitation of space, we provide a detailed discuss of related works in Appendix B.

We leverage deep learning methods to improve the approximation of NE of the inexact solvers. One straightforward method is supervised learning (SL) [Krizhevsky *et al.*, 2012; He *et al.*, 2016], i.e., training a deep neural network which takes the game as input and outputs the NE directly. However, there are several issues of SL methods. First, it is difficult to generate the high quality dataset for SL methods due to: i) computing NE as labels is time-consuming and there are no efficient methods for computing NE, thus the generation of training datasets in SL methods is inefficient, ii) one game may have multiple NEs, i.e., multiple labels, and the computation of all equilibria and the equilibrium selection problem is not addressed [Harsanyi *et al.*, 1988]. Second, the SL methods will inevitably generate the inexact solutions and further refinement methods are required to improve the approximation results, which makes the methods to be complicated. Therefore, inspired by [Wang *et al.*, 2021], we leverage reinforcement learning (RL) methods, instead of using SL methods, to achieve this goal, i.e., *iteratively modifying the games with different sizes with a single strategy learned by RL methods and applying the solvers, e.g., $\alpha$-rank, to the modified games to obtain the solutions which are evaluated on the original games*. However, there are two main issues: i) the games with different sizes result in varied inputs to the RL methods, where a simple neural network architecture, e.g., multi-layer perceptron (MLP) cannot handle, and ii) the number of payoff values grows exponentially with the number of players and actions, which brings difficulties to modify the games if we only change a payoff value once.

To address the above issues, we propose REinforcement Nash Equilibrium Solver (RENES). Our main contributions are three-fold. First, we represent the games with different sizes as $\alpha$-rank response graphs, which are used to characterize the intrinsic properties of games [Omidshafiei *et al.*, 2020], and then leverage the graph neural network (GNN) to take the $\alpha$-rank response graphs as inputs. Second, we use tensor decomposition, e.g., canonical polyadic (CP), to make the modifying actions fixed for games with different sizes, rather than changing a payoff value once. Third, we train the modifying strategy for games with the widely-used proximal policy optimization (PPO) and apply the solvers to solve the modified games, where the obtained solution is evaluated on original games. Extensive experiments on large-scale normal-form games, i.e., 3000 sampled games for training and 500 sampled games for testing, show that our method can further improve the approximation of NE of different solvers, i.e., $\alpha$-rank, CE, FP and PRD, and can be generalized to unseen games. To the best of our current knowledge, this work is the first effort in game theory that leverages RL methods to train a single strategy for modifying the games, and so as to improve the solvers' approximation performances.

## 2 Preliminaries

We present the preliminaries of game theory in this section.

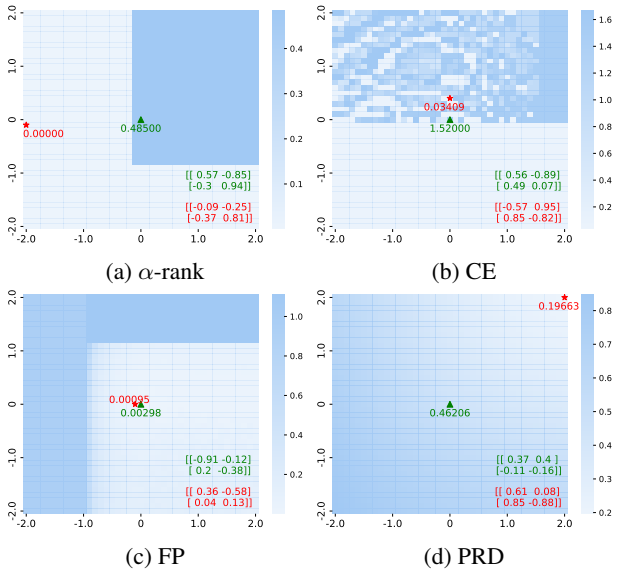
**Normal-form Games.** Consider the $K$-player normal-form game, where each player $k \in [K]$ has a finite set of actions $\mathcal{A}^k$. We use $\mathcal{A}^{-k}$ to represent the action space excluding the player $k$, also for other terms. We denote the joint action space as $\mathcal{A} = \times_{k\in[K]}\mathcal{A}^k$. Let $\boldsymbol{a} \in \mathcal{A}$ be the joint action of $K$ players and $M(\boldsymbol{a}) = \langle M^k(\boldsymbol{a})\rangle \in \mathbb{R}^K$ is the payoff vector of players when playing the action $\boldsymbol{a}$. A mixed strategy profile is defined as $\pi \in \Delta(\mathcal{A})$, which is a distribution over $\mathcal{A}$ and $\pi(\boldsymbol{a})$ is the probability that the joint action $\boldsymbol{a}$ will be played. The expected payoff of player $k \in [K]$ is denoted as $M^k(\pi) = \sum_{\boldsymbol{a}\in\mathcal{A}} \pi(\boldsymbol{a})M^k(\boldsymbol{a})$.

**Solution Concepts.** Given a mixed strategy $\pi$, the best response of player $k \in [K]$ is defined as $\text{BR}^k(\pi) = \arg\max_{\mu\in\Delta(\mathcal{A}^k)}[M^k(\mu, \pi^{-k})]$. A factorized mixed strategy $\pi(\boldsymbol{a}) = \prod_{k\in[K]} \pi^k(a^k)$ is Nash Equilibrium (NE) if $\pi^k \in \text{BR}^k(\pi)$ for $k \in [K]$. We define the NashConv value as $\texttt{NC}\,(\pi) = \sum_{k\in[K]} M^k(\text{BR}^k(\pi), \pi^{-k}) - M^k(\pi)$ to measure the distance of the mixed strategy from an NE. Computing NE in general-sum games is PPAD-Complete [Daskalakis *et al.*, 2009]. Therefore, many alternative solutions are proposed to approximate NE. One of the most investigated alternative solutions is Correlated Equilibrium (CE) [Aumann, 1987]. A mixed strategy profile is CE if for all $k \in [K]$,

$$\sum_{\boldsymbol{a}^{-k}\in\pi^{-k}} \pi(\boldsymbol{a})[M^k(\boldsymbol{a}) - M^k(\boldsymbol{a}^{-k}, b^k)] \geq 0 \qquad (1)$$

where $b^k \neq a^k$ and $b^k \in \mathcal{A}^k$. Any NE is a CE, therefore, CE is a generalized solution concept of NE, and CE is equivalent to NE in two-player zero-sum games. CE can be computed in polynomial time and many learning methods, e.g., regret matching [Hart and Mas-Colell, 2000], can lead to CE. As any NE is CE, CE also suffers the equilibrium problem, therefore, researchers propose many measures to select the equilibrium, e.g., maximum welfare CE and maximum entropy CE [Marris *et al.*, 2021]. $\alpha$-rank is recently proposed in [Omidshafiei *et al.*, 2019], which is the stationary distribution of the *$\alpha$-rank response graph* constructed from the game. We will introduce the $\alpha$-rank response graph, as well as $\alpha$-rank in detail, in Section 4.1 as we also use the $\alpha$-rank response graph to transform the games into graphs.

**Learning Methods.** Instead of considering alternative solutions, many learning methods are proposed to approximate NE. Fictitious play (FP) [Brown, 1951; Heinrich *et al.*, 2015] is a famous method to approximate NE. FP starts with arbitrary strategies for players, and at each round, each player will compute the best response to the opponents' average behaviors. FP can converge to NE in certain classes of games, e.g., two-player zero-sum and many-player potential games [Monderer and Shapley, 1996], while convergence is not guaranteed in




(a) $\alpha$-rank (b) CE

(c) FP (d) PRD

Figure 1: Motivating Examples. The payoff matrices of the two players are displayed in the figure in green and red colors, respectively. The x-axis and y-axis are the values of $\delta^1$ and $\delta^2$, respectively. For plotting, the interval $[-2, 2]$ is discredited with a step size 0.1. The NashConv values of the solvers on the original games $M$ is marked as the green triangle and the minimal NashConv value is marked as the red star. We note that the games $M$ selected for the solvers are specifically designed and differ from each other.

general games. Replicator dynamics (RD) [Schuster and Sigmund, 1983] is a learning dynamics from evolutionary game theory, which is defined as

$$\dot{\pi}_t^k(a^k) = \pi_t^k(a^k)[M^k(a^k, \pi_t^{-k}) - M^k(\pi_t)], \qquad (2)$$

and RD can converge to NE under certain conditions. To ensure the exploration of RD, a variant of RD, projected replicator dynamics (PRD) is proposed in [Lanctot *et al.*, 2017], which projects the strategy to the set $\Delta^\gamma(\mathcal{A}^k) = \{\pi^k \in \Delta(\mathcal{A}^k) | \pi^k(a^k) \geq \frac{\gamma}{|\mathcal{A}^k|+1}, \forall a^k \in \mathcal{A}^k\}$. PRD is demonstrated to be more effective than RD to approximate NE in many games empirically [Lanctot *et al.*, 2017; Li and Wellman, 2021].

## 3 Motivating Examples

In this section, we provide motivating examples of RENES to demonstrate that the strategies obtained by alternative solutions and learning methods can diverge from NE strategies and can be improve through modifying the games.

Specifically, we consider the general -sum games with 2 players and each player has 2 actions. Thus, the game can be represented as a $2 \times 2 \times 2$ tensor $M = \langle M^1, M^2 \rangle$ where $M^k = \begin{bmatrix} m_{11}^k & m_{12}^k \\ m_{21}^k & m_{22}^k \end{bmatrix}$, $k \in \{1, 2\}$ and the payoff values are in $[-1, 1]$. For simplicity and easy visualization, we only modify the payoff values of the first actions of players. Denoting the modification values as $\delta^1$ and $\delta^2$, therefore, the modified game is $\tilde{M} = \langle \tilde{M}^1, \tilde{M}^2 \rangle$ where $\tilde{M}^k = \begin{bmatrix} m_{11}^k + \delta^k & m_{12}^k \\ m_{21}^k & m_{22}^k \end{bmatrix}$, $k \in \{1, 2\}$. The values of $\delta^k \in [-2, 2]$. We then apply the considered solvers, i.e., alternative solutions and learning methods, to the modified games and evaluate on the original games.

The results are displayed in Figure 1. We can observe that in the specifically designed games, the solvers diverge from the exact NE. Through only modifying the payoff values of the first actions, we can significantly improve the approximation of $\alpha$-rank, CE and PRD and also slightly improve on FP. With larger spaces of the modification, we can even further improve the approximation of solvers. The results in Figure 1 demonstrate the effectiveness of the fundamental idea of RENES, i.e., improving the approximation of solvers through modifying the games. However, there are still several issues: i) when the space of modification is larger, the enumeration of all possible combinations is impossible, ii) for different games with different sizes, the modification spaces vary significantly. Therefore, we propose RENES which can provide the efficient modification strategies for the games with different sizes.

## 4 RENES

In this section, we introduce the proposed REinforcement Nash Equilibrium Solver (RENES). The general procedure of RENES is displayed in Figure 2. Specifically, RENES is formed with three components: i) a modification oracle $\mathcal{O}$, represented as neural networks and trained with RL methods, e.g., PPO [Schulman *et al.*, 2017], which takes the original game $M$ as input and generates the modified game $M'$, ii) an inexact solver $\mathcal{H}$, which takes the modified game $M'$ and generates the solution $\pi$, and iii) an evaluation measure $\mathcal{E}$, which evaluates the obtained solution $\pi$ on the original game $M$ and provides the reward signal to the RL method when training the modification oracle $\mathcal{O}$. The solvers considered in this paper are $\alpha$-rank, CE, FP, PRD and the evaluation measure used in this paper is NashConv [Muller *et al.*, 2020]. We will provide the details of the design and training of the modification oracle $\mathcal{O}$ in the rest of this section.

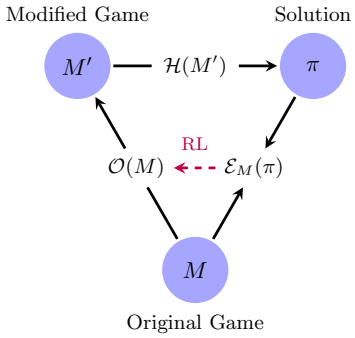



Figure 2: Flow of RENES. Specifically, staring with the original game $M$, the modification oracle $\mathcal{O}$ modified the game to $M'$ and the solver $\mathcal{H}$ is applied to the modified game $M'$. The obtained solution $\pi$ is evaluated on $M$ with $\mathcal{E}$.

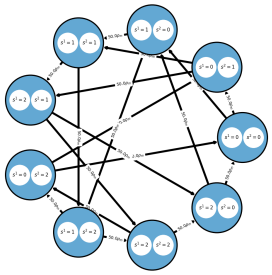
### 4.1 Games as $\alpha$-Rank Response Graphs

The normal-form games are normally represented as high-dimensional tensors. However, current deep learning methods cannot efficiently handle high dimensional tensors. There

are two possible solutions: i) flattening the tensor into a 1-D vector and using MLP as the modification oracle $\mathcal{O}$, where the relations between payoff values will be eliminated and the length of the vector is varied for games with different sizes, and ii) using convolutional neural network (CNN), which is efficient to process images [He *et al.*, 2016]. However, CNN still cannot process high dimensional tensors, e.g., 4-D, and cannot handle games with different sizes. Another option is that we can add a cap of the game size and use the maximum size of the game to build the policy network (we use 0 to fill in the tensor if the game is smaller than the maximum size during training and testing), however, this will still hurt the generalizability of RENES, as it cannot handle the games beyond the maximum size.

| | R | P | S |
|---|---|---|---|
| R | 0 | $-1$ | 1 |
| P | 1 | 0 | $-1$ |
| S | $-1$ | 1 | 0 |

(a) Payoff table



(b) $\alpha$-rank response graph

Figure 3: Payoff table and $\alpha$-rank response graph for Rock-Paper-Scissors (RPS) game, where $\alpha = 100$ and $m = 50$ when computing the $\alpha$-rank response graph.

To handle the games with different sizes, we represent the games as the $\alpha$-rank response graphs, which is shown to represent the intrinsic properties of games in [Omidshafiei *et al.*, 2020], and then use graph neural network (GNN) [Kipf and Welling, 2016; Veličković *et al.*, 2018; Yun *et al.*, 2019] to extract the features of games. We note that GNN can efficiently handle the graphs with different sizes [Li *et al.*, 2018], as it takes the neighboring information to update the node embeddings. The definition of $\alpha$-rank response graph, as well as $\alpha$-rank, is defined in Definition 1, where each joint action corresponds to a node on the graph, and the two nodes are connected if there is only one player has different action in the joint actions. The transition probabilities are determined by Eq. (3). A concrete example of $\alpha$-rank response graph for the rock-paper-scissors game is displayed in Figure 3. After representing the games as graphs, we use the same value, e.g., 1.0, as the node features and the transition probabilities as the weights of edges. Then, we use GNN, e.g., GCN [Kipf and Welling, 2016] to process the $\alpha$-rank response graph to obtain the embedding of the graph, and then we add an MLP to generate the outputs. We use PPO to train the modification oracle, where both actor and critic use this architecture to generate the action and the state-action value, respectively. More training details can be found in Section 4.4.

**Definition 1.** *The $\alpha$-rank response graph is defined over all joint pure actions, specified by a transition matrix $C$. Given a joint pure action $\boldsymbol{a}$ and $\boldsymbol{a}' = \langle \sigma^k, \boldsymbol{a}^{-k} \rangle$ is a joint pure action which equals to $\boldsymbol{a}$ except the player $k$, we denote the transition probability from $\boldsymbol{a}$ to $\boldsymbol{a}'$ as $C(\boldsymbol{a}, \boldsymbol{a}')$*

$$\begin{cases} \eta \frac{1-\exp(-\alpha(M^k(\boldsymbol{a}')-M^k(\boldsymbol{a})))}{1-\exp(-\alpha m(M^k(\boldsymbol{a}')-M^k(\boldsymbol{a})))}, & \text{if } M^k(\boldsymbol{a}) \neq M^k(\boldsymbol{a}') \\ \frac{\eta}{m}, & \text{otherwise} \end{cases} \tag{3}$$

*where $\eta = 1/(\sum_{k\in[K]}(|\mathcal{A}^k| - 1))$. The self-transition probability is defined as $C(\boldsymbol{a}, \boldsymbol{a}) = 1 - \sum_{k\in[k],\boldsymbol{a}'|\sigma^k\in\mathcal{A}^k} C(\boldsymbol{a}, \boldsymbol{a}')$. If two joint actions $\boldsymbol{a}$ and $\boldsymbol{a}'$ differ in more than one player's action, $C(\boldsymbol{a}, \boldsymbol{a}') = 0$. The values of $\alpha$ and $m$ are the selection pressure and the number of populations, respectively, which are specified by users. The transition matrix $C$ define a directed graph, i.e., $\alpha$-rank response graph, where the stationary distribution of $C$ is the $\alpha$-rank distribution.*

## 4.2 Action Spaces of RENES

Most RL methods are designed for the problems with fixed action spaces. However, when the game sizes change, the number of elements which can be changed is also different. Therefore, the action space of RENES needs to be specially designed. A naive definition of the action space is that at each step, RENES changes the payoff of a player in a specific joint action, i.e., generating the indices of the elements in the payoff table and the way to modify this payoff value. In this case, the action size of RENES is $K \cdot \prod_{k\in[K]} |\mathcal{A}^k|$, which grows exponentially along with the number of actions of each player. On the other hand, we still need to specify the maximum size of the game sizes to generate valid indices of the elements, which hurts the generalizability of RENES.

Therefore, we consider a more compact action space with tensor decomposition [Kolda and Bader, 2009]. Specifically, we use the canonical polyadic (CP) decomposition of the payoff table $M$ and set the rank $r$ to be fixed and the action of RENES is the coefficients over $r$:

$$M \approx \sum_{i=1}^{r} \lambda_i \cdot m_{1,i} \otimes m_{2,i} \otimes \cdots \otimes m_{K+1,i}, \tag{4}$$

where $\boldsymbol{\lambda} = \langle \lambda_i \rangle, i = 1, \cdots, r$ are the weights of the decomposed tensors and $m_{k,i}, k \in \{1, \ldots, K+1\}$ are the factors which are used to modify the game. For the decomposition, the weight $\boldsymbol{\lambda} = \mathbf{1}$[1]. Given any arbitrary weight $\boldsymbol{\lambda}$, we can reconstruct the payoff tensor with the reconstruction oracle $\mathcal{R}_M(\boldsymbol{\lambda})$. Therefore, we let the modified oracle $\mathcal{O}$ to modify the weights and update the game by

$$M_t = M_{t-1} + \eta \cdot \mathcal{R}_M(\boldsymbol{\lambda}). \tag{5}$$

With the tensor decomposition, we can use a fixed size of action space of RENES, specified by $r$. The tensor decomposition can be viewed as a simple method of the abstraction [Brown and Sandholm, 2015; Brown and Sandholm, 2017], and more sophisticated and decomposition methods can be considered in future works [Burch *et al.*, 2014].

## 4.3 MDP Formulation of RENES

As RL relies on the Markov Decision Process (MDP) formulation [Schulman *et al.*, 2017], we will also first reformulate our problem into an MDP:

[1]The tensor decomposition is implemented by `TensorLy` (https://github.com/tensorly/tensorly). Other implemented decomposition methods can also be used.

- **States.** The state of RENES is defined as the tuple with the original game $M_0$ and the current game $M_{t-1}$. Note that we compute the solution with the current game $M_{t-1}$ and evaluate the obtained solution on the original game $M_0$, therefore, $\langle M_0, M_{t-1}\rangle$ give the full information of the underlying MDP to solve, therefore, we ignore all the intermediate games generated, i.e., $M_t, t \in \{1, \dots, t-2\}$, which can simplify and stabilize the training.
- **Actions.** The actions of RENES are the weights $\boldsymbol{\lambda}$ over the decomposition factors, which will be used to update $M_{t-1}$ to obtain $M_t$ following Eq. (5).
- **Transition Function.** After obtaining the new modified game $M_t$, the problem will transit to the new state defined by the tuple of $M_0$ and $M_t$.
- **Reward Function.** The immediate reward at step $t$ is defined as $\texttt{NC}(\pi_{t-1}) - \texttt{NC}(\pi_t)$, where $\pi_t$ is the strategy obtained when applying the solver to the modified game $M_t$. As we use NashConv (lower is better) as our evaluation measure, the decrement of the NashConv will be the positive reward of RENES. With maximizing the accumulated reward, RENES can boost the approximation of NE of the solver. We also consider using a normalized NashConv measure, defined as $[\texttt{NC}(\pi_{t-1}) - \texttt{NC}(\pi_t)]/\texttt{NC}(\pi_0)$, where $\texttt{NC}(\pi_0)$ is the normalizer. This normalized NashConv can make the performances on different games comparable. However, when $\texttt{NC}(\pi_0)$ is small, e.g., less than $0.01$, and $|\texttt{NC}(\pi_{t-1}) - \texttt{NC}(\pi_t)|$ is much larger than $\texttt{NC}(\pi_0)$, e.g., $1.0$, this can make the reward value be extremely large and cannot be used for training. Therefore, the normalized NashConv is only used for evaluation.
- **Horizon & Discounting Factor.** The horizon $T$ specifies the maximum step, e.g., 50, where the modification can be used to modify the game for better performance. The discounting factor is denoted as $\gamma \in (0, 1]$.

### 4.4 Training RENES with PPO

After reformulating the training of RENES as MDP, we then train RENES with RL methods. RL is an area of the policy optimization in complex sequential decision-making environments [Sutton and Barto, 2018]. RL methods rely on the trail-and-error process to explore the solution space for better policies. The primary RL method is Q-learning [Watkins and Dayan, 1992; Mnih *et al.*, 2015], which can only be used on the problems with discrete actions, and the policy gradient methods are proposed for the problems with both discrete and continuous actions [Sutton *et al.*, 1999; Mnih *et al.*, 2016; Haarnoja *et al.*, 2018].

PPO is an on-policy policy gradient method, which is a simplified, but more data efficient and reliable, variant of Trust Region Policy Optimization (TRPO) [Schulman *et al.*, 2015], which leverages the "trust region" to bound the update of the policy to avoid training collapse. Compared with TRPO, PPO is more data efficient and with more reliable performances than TRPO, while only using the first-order optimization for computational efficiency.

Specifically, PPO is maximizing the objective

$$J(\theta) = \mathbb{E}\left[\min(r_\theta \cdot A, \texttt{clip}(r_\theta, 1-\epsilon, 1+\epsilon) \cdot A)\right], \quad (6)$$

where $r_\theta$ is the importance sampling ratio conditional on $\theta$, $\theta$ is the parameter of the policy, $A$ is the advantage value which is computed by using the discounted accumulative reward minus the critic network prediction of the state-action value, and $\epsilon$ is the hyperparameter which controls the boundary of the trust region. We only provide a short introduction of PPO in this section, as we take PPO as a blackbox for optimizing the modification oracle $\mathcal{O}$. Other RL methods, e.g., soft actor critic (SAC) [Haarnoja *et al.*, 2018], can be used and for more details of RL, we refer readers to [Sutton and Barto, 2018].

## 5 Experiments

In this section, we present the experimental results of RENES on large-scale normal-form games. We consider two cases: i) **simple case** where all games have the same size to verify the idea of modifying the games to boost the performance of inexact solvers, and ii) **general case** where the games have different sizes to verify that the design of RENES can handle the game with different sizes. We then conduct the ablation study on the numbers of the action dimensions and the horizon.

### 5.1 Experimental Setups

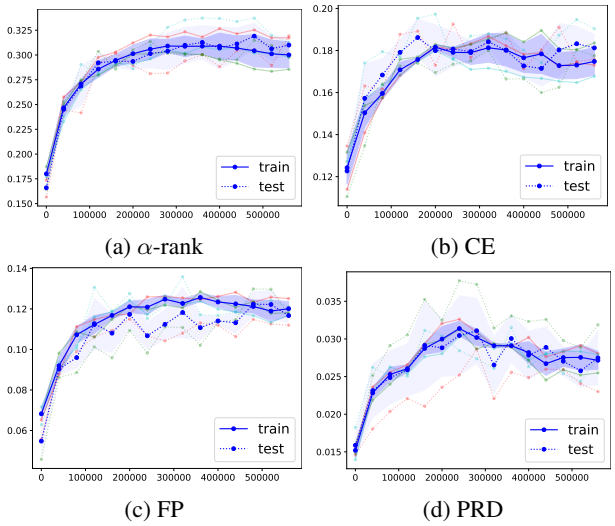
**Games and Solvers.** For the games, we randomly sample 3000 games for training, and 500 games for testing to verify the ability of Renes to generalize to unseen games. The games are represented as high dimensional payoff tensors with dimensions as $[K, |\mathcal{A}^1|, \dots, |\mathcal{A}^K|]$, which are used for tensor decomposition. We also do the normalization of the payoff to stablize the training. The four selected solvers are: $\alpha$-rank, CE, FP and PRD. For $\alpha$-rank and PRD, we use the implementations in OpenSpiel [Lanctot *et al.*, 2019][2]. We use regret matching [Hart and Mas-Colell, 2000] in OpenSpiel for CE and implement FP by ourselves. More details of the games and solvers are in Appendix B.

**Evaluation Measures.** Different from RL where the accumulated reward is considered, we focus on the minimum NashConv during the $T$ steps of the modification. As the solvers have different performances on different games, we would use the normalized NashConv measure to make the performances comparable. Specifically, the performance of RENES on a specific game is measured by $1 - \frac{\min_{t\in[T]}\{\texttt{NC}(\pi_t)\}}{\texttt{NC}(\pi_0)}$, which measures the relative improvement over the performance of the solver on the original game, therefore, 0 implies no improvement and 1 implies the obtained solution is the exact NE. We take the average over all games to measure the performance.

**Baselines.** We take the two baselines to compare with RENES: i) the performance of the solver on the original games without any modification, ii) the performance of the random policy of modifying games. The two baselines correspond to the value of 0 and the performance at 0 step of the RL training using the proposed measure. Most of the previous methods require running the methods on specific games and the learned policies cannot be generalized to other games, while RENES can generalize to unseen games without training, so we do not consider these SoTA methods as our baselines.[3]

[2] https://github.com/deepmind/open_spiel

[3] More justifications of the baselines can be found in Appendix A.




(a) $\alpha$-rank (b) CE

(c) FP (d) PRD

Figure 4: Results of RENES in simple case. The solid lines and dotted lines are the results on the training set and the testing set of games, respectively. The transparent lines are the results with different seeds, as the runs will different seeds achieve the best performances in different epochs, so we plot them for better understanding of the training across different seeds. The blue line is the averaged results and the shaded area plots the standard deviation. Note that the y-axis scale differs across figures for better visualizations. The same style is also adopted in Figures 5 and 6.

**Training with PPO** For the training, we set the decomposition rank $r = 10$, i.e., the number of the action dimensions is 10, and $T = 50$, i.e., the number of the maximum steps of the modification is 50. Conceptually, the more action dimensions and the more steps, the better performance of RENES. Larger values of $r$ may bring the difficulties of training and make the training unstable. As at each time step, we need to run solvers to solve the modified game, which is time-consuming when $T$ is very large. We choose these two values to balance the trade-off between the performance and the efficiency. We run the experiments with three seeds. The values of the hyperparameters can be found in Appendix B. Due to the limitation of the computational resources, we do not conduct the exhaustive tuning of the hyperparameters.

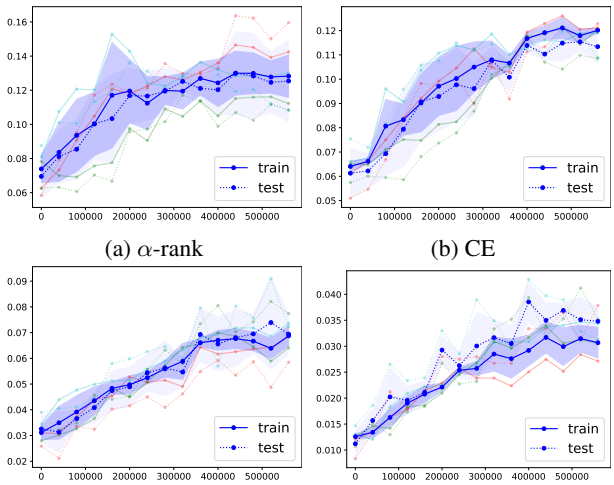
## 5.2 Simple Case

In this section, we present the experiments on the simple case. For the simple case, we randomly sample 3000 games for training and 500 games for testing where all games are 2-player games with 5 actions of each player. As all games have the same size, we simply flatten the payoff tensors to 1-D vectors and using MLP for the policy and critic in PPO.

The results of the simple case are displayed in Figure 4 and Table 1. We observe that the training on the training set of games can be generalized to the testing set, which indicates that RENES can be used as a general policy to modify the games, even on unseen games. More specifically, we observe that RENES can significantly boost the performance of $\alpha$-rank, i.e., larger than $0.3$ over all three seeds, as shown in Figure 4a, and achieve $0.313$ and $0.324$ on training and testing sets, respectively. For PRD, both random policy and RENES can only bring small improvements, i.e., smaller than $0.05$ over all seeds, as shown in Figure 4d, and only achieve $0.032$ and $0.033$ on training and testing sets, respectively. For the other solvers, RENES can bring notable improvements, i.e., larger than $0.16$ for CE and $0.1$ for FP over three seeds. Overall, we can conclude that the performances of inexact solvers can be boosted through modifying the games. We also observe that longer training of RENES does not necessarily improve the performance, i.e., Figures 4a, 4c and 4d, as observed in other RL experiments. We believe that with more tuning of the hyperparameters, RENES can achieve better performances.

## 5.3 General Case

We then conduct the experiments on the general case. In the general case, we also sample 3000 games for training and 500 games for testing, where the games have $\{2, 3\}$ players and each player is with $\{2, 3, 4\}$ actions. We only focus on small games as the running of solvers for large multi-player is even more time-consuming. As the game sizes vary in this case, we will take the GNN as the base network and add an MLP head to form the policy and the critic in PPO, respectively. Other settings are the same as the simple case.




(a) $\alpha$-rank (b) CE

(c) FP (d) PRD

Figure 5: Results of RENES in general case

The results of the general case are displayed in Figure 5 and Table 2. We observe that similar to the simple case, the policy of RENES trained on the training set can be generalized to the testing set, which indicates that RENES can be a general policy for unseen games even when the game sizes vary. Compared with the simple case, RENES achieves lower performances over different solvers on the general case. Specifically, RENES still brings the largest improvement for $\alpha$-rank, i.e., $0.139$ on both training and testing dataset, and the smallest improvement for PRD, i.e., $0.032$ and $0.041$ on training and test datasets, respectively. For the other two solvers, RENES also brings notable improvements, i.e., larger than $0.120$ and $0.071$ on both training and test datasets, respectively.

To summarize, for both simple and general cases, we observe that RENES can improve the approximations of NE for

| | | $\alpha$-rank | CE | FP | PRD |
|---|---|---|---|---|---|
| Training | Random | 0.180(0.006) | 0.124(0.007) | 0.068(0.003) | 0.015(0.001) |
| | RENES | 0.313(0.010) | 0.185(0.004) | 0.128(0.001) | 0.032(0.001) |
| Testing | Random | 0.180(0.006) | 0.123(0.010) | 0.055(0.007) | 0.016(0.002) |
| | RENES | 0.324(0.010) | 0.190(0.007) | 0.127(0.009) | 0.033(0.004) |

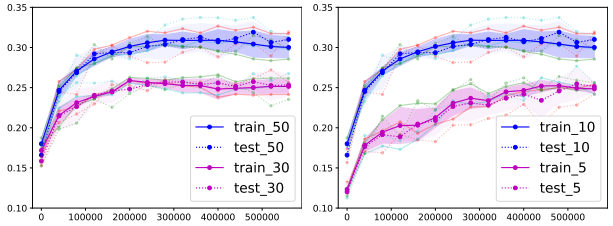
Table 1: Results of RENES in Simple Case. To calculate the values, we pick the best values across different epochs for each seed and compute the mean values and standard deviation values.

| | | $\alpha$-rank | CE | FP | PRD |
|---|---|---|---|---|---|
| Training | Random | 0.074(0.008) | 0.064(0.002) | 0.031(0.003) | 0.013(0.001) |
| | RENES | 0.139(0.016) | 0.122(0.003) | 0.071(0.002) | 0.032(0.003) |
| Testing | Random | 0.074(0.005) | 0.061(0.010) | 0.033(0.005) | 0.011(0.003) |
| | RENES | 0.139(0.019) | 0.120(0.002) | 0.077(0.013) | 0.041(0.002) |

Table 2: Results of RENES in General Case.

different existing solvers, i.e., $\alpha$-rank, CE, FP and PRD. We believe that RENES can be an orthogonal tool for approximating NE in multi-player general-sum games. With combining the advanced solvers and RENES, we can further achieve a better approximation of NE.

### 5.4 Ablations




(a) Horizon $T$ (b) Action dimensions $r$

Figure 6: Ablations on $\alpha$-rank in simple case.

We present the ablation results in this section. We ablate two configurations determined by us: i) the number of maximum steps, i.e., horizon $T$, and ii) the number of the action dimension $r$. These two configurations are the main hyperparameters of RENES related to game theory. And for other hyperparameters related to PPO, we do not conduct the ablations. For efficiency, the ablation experiments are conducted on the simple case with $\alpha$-rank as the solver.

The ablation results are displayed in Figure 6. We observe that with more steps and more actions, the performances are better, which indicates that there is a trade-off between the performance and the efficiency of RENES, and advanced methods can be used for the optimal configurations of the two values given the limited resources, e.g., Optuna [Akiba *et al.*, 2019]. We also observe that when the horizon is longer and the number of action dimensions is larger, the results are more sensitive to the seeds, which may due to the intrinsic of the randomness of the initialization of PPO.

## 6 Discussion

### 6.1 Limitations and Future Works

The limitations of RENES are as follows: i) We only conduct experiments on small normal-form games, as for larger games, the running time of RENES, as well as the solvers, will increase. We will consider scaling RENES up to large normal-form games, e.g., 5 players and 30 actions each player, in the future. ii) We only conduct experiments on normal-form games, while extensive-form games (EFGs) are more difficult for computing NE and novel methods to modifying EFGs are required, i.e., the methods to handle the imperfect information and the sequential properties in EFGs. iii) We only focus on NE in this paper. The core idea of modifying the games to facilitate the computation can also be generalized to other solution concepts, e.g., quantal response equilibrium (QRE) [McKelvey and Palfrey, 1995]. Due to the space limitation, we provide a detailed discussion about the limitations and future works in Appendix C.

### 6.2 Conclusion

In this work, we propose RENES, which leverages RL methods to find a single policy to modify the original games with different sizes and applies existing solvers to solve the modified games. Our contributions are threefold: i) We adopt the $\alpha$-rank response graph as the representation of the game to make RENES handle the games with different sizes; ii) We leverage the tensor decomposition to improve the efficiency of the modification; iii) We train RENES with the widely-used PPO method. Extensive experiments show that our method can boost the performances of solvers and generate more accurate approximation of NE. To the best of our knowledge, this work is the first attempt to leverage RL methods to train a single policy to modify the games to improve the approximation performances of different solvers in game theory. We hope this method can open a new venue of modifying games as the pre-training task, complimentary to new methods and solutions, to approximate NE with generalizability and efficiency.

## Acknowledgements

This research is supported by the National Research Foundation Singapore and DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2-GC-2023-009). Hau Chan is supported by the National Institute of General Medical Sciences of the National Institutes of Health [P20GM130461], the Rural Drug Addiction Research Center at the University of Nebraska-Lincoln, and the National Science Foundation under grant IIS:RI #2302999. Any opinions, findings and conclusions, or recommendations expressed in this material are those of the author(s) and do not reflect the views of the funding agencies.

## Contribution Statement

Xinrun Wang and Chang Yang are the co-first and co-corresponding authors who contribute to this work equally.

## References

[Akiba *et al.*, 2019] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *KDD*, pages 2623–2631, 2019.

[Aumann, 1974] Robert J Aumann. Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*, 1(1):67–96, 1974.

[Aumann, 1987] Robert J Aumann. Correlated equilibrium as an expression of Bayesian rationality. *Econometrica: Journal of the Econometric Society*, pages 1–18, 1987.

[Brown and Sandholm, 2015] Noam Brown and Tuomas Sandholm. Simultaneous abstraction and equilibrium finding in games. In *IJCAI*, pages 489–496, 2015.

[Brown and Sandholm, 2017] Noam Brown and Tuomas Sandholm. Safe and nested subgame solving for imperfect-information games. In *NeurIPS*, pages 689–699, 2017.

[Brown, 1951] George W Brown. Iterative solution of games by fictitious play. *Activity Analysis of Production and Allocation*, 13(1):374, 1951.

[Burch *et al.*, 2014] Neil Burch, Michael Johanson, and Michael Bowling. Solving imperfect information games using decomposition. In *AAAI*, 2014.

[Chen *et al.*, 2009] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56(3):1–57, 2009.

[Daskalakis *et al.*, 2009] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

[Fudenberg and Tirole, 1991] Drew Fudenberg and Jean Tirole. *Game Theory*. MIT press, 1991.

[Gemp *et al.*, 2022] Ian Gemp, Rahul Savani, Marc Lanctot, Yoram Bachrach, Thomas Anthony, Richard Everett, Andrea Tacchetti, Tom Eccles, and János Kramár. Sample-based approximation of nash in large many-player games via gradient descent. In *AAMAS*, pages 507–515, 2022.

[Goldberg *et al.*, 2013] Paul W Goldberg, Christos H Papadimitriou, and Rahul Savani. The complexity of the homotopy method, equilibrium selection, and Lemke-Howson solutions. *ACM Transactions on Economics and Computation (TEAC)*, 1(2):1–25, 2013.

[Haarnoja *et al.*, 2018] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, pages 1861–1870, 2018.

[Harsanyi *et al.*, 1988] John C Harsanyi, Reinhard Selten, et al. A general theory of equilibrium selection in games. *MIT Press Books*, 1, 1988.

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

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[Heinrich *et al.*, 2015] Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *ICML*, pages 805–813, 2015.

[Herings and Peeters, 2010] P Herings and Ronald Peeters. Homotopy methods to compute equilibria in game theory. *Economic Theory*, 42(1):119–156, 2010.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[Kolda and Bader, 2009] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[Lanctot *et al.*, 2017] Marc Lanctot, Vinicius Zambaldi, Audrūnas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *NeurIPS*, pages 4193–4206, 2017.

[Lanctot *et al.*, 2019] Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, et al. OpenSpiel: A framework for reinforcement learning in games. *arXiv preprint arXiv:1908.09453*, 2019.

[Lemke and Howson, 1964] Carlton E Lemke and Joseph T Howson, Jr. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):413–423, 1964.

[Li and Wellman, 2021] Zun Li and Michael P Wellman. Evolution strategies for approximate solution of bayesian games. In *AAAI*, pages 5531–5540, 2021.

[Li *et al.*, 2018] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *NeurIPS*, pages 537–546, 2018.

[Marris *et al.*, 2021] Luke Marris, Paul Muller, Marc Lanctot, Karl Tuyls, and Thore Graepel. Multi-agent training beyond zero-sum with correlated equilibrium meta-solvers. In *ICML*, pages 7480–7491, 2021.

[McKelvey and Palfrey, 1995] Richard D McKelvey and Thomas R Palfrey. Quantal response equilibria for normal form games. *Games and economic behavior*, 10(1):6–38, 1995.

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

[Mnih *et al.*, 2016] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, pages 1928–1937, 2016.

[Monderer and Shapley, 1996] Dov Monderer and Lloyd S Shapley. Fictitious play property for games with identical interests. *Journal of Economic Theory*, 68(1):258–265, 1996.

[Muller *et al.*, 2020] Paul Muller, Shayegan Omidshafiei, Mark Rowland, Karl Tuyls, Julien Perolat, Siqi Liu, Daniel Hennes, Luke Marris, Marc Lanctot, Edward Hughes, et al. A generalized training approach for multiagent learning. In *ICLR*, 2020.

[Myerson, 1999] Roger B Myerson. Nash equilibrium and the history of economic theory. *Journal of Economic Literature*, 37(3):1067–1082, 1999.

[Nash Jr, 1950] John F Nash Jr. Equilibrium points in n-person games. *PNAS*, 36(1):48–49, 1950.

[Omidshafiei *et al.*, 2019] Shayegan Omidshafiei, Christos Papadimitriou, Georgios Piliouras, Karl Tuyls, Mark Rowland, Jean-Baptiste Lespiau, Wojciech M Czarnecki, Marc Lanctot, Julien Perolat, and Remi Munos. $\alpha$-rank: Multi-agent evaluation by evolution. *Scientific Reports*, 9(1):1–29, 2019.

[Omidshafiei *et al.*, 2020] Shayegan Omidshafiei, Karl Tuyls, Wojciech M Czarnecki, Francisco C Santos, Mark Rowland, Jerome Connor, Daniel Hennes, Paul Muller, Julien Pérolat, Bart De Vylder, et al. Navigating the landscape of multiplayer games. *Nature Communications*, 11(1):1–17, 2020.

[Schulman *et al.*, 2015] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, pages 1889–1897, 2015.

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

[Schuster and Sigmund, 1983] Peter Schuster and Karl Sigmund. Replicator dynamics. *Journal of Theoretical Biology*, 100(3):533–538, 1983.

[Sutton and Barto, 2018] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

[Sutton *et al.*, 1999] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS*, pages 1057–1063, 1999.

[Taylor and Jonker, 1978] Peter D Taylor and Leo B Jonker. Evolutionary stable strategies and game dynamics. *Mathematical biosciences*, 40(1-2):145–156, 1978.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[Wang *et al.*, 2021] Runzhong Wang, Zhigang Hua, Gan Liu, Jiayi Zhang, Junchi Yan, Feng Qi, Shuang Yang, Jun Zhou, and Xiaokang Yang. A bi-level framework for learning to solve combinatorial optimization on graphs. In *NeurIPS*, 2021.

[Watkins and Dayan, 1992] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[Yun *et al.*, 2019] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. In *NeurIPS*, pages 11983–11993, 2019.