Solving Large-Scale Pursuit-Evasion Games Using Pre-trained Strategies

Shuxin Li[1], Xinrun Wang[1]*, Youzhi Zhang[2]*, Wanqi Xue[1], Jakub Černý[1], Bo An[1]

[1] School of Computer Science and Engineering, Nanyang Technological University, Singapore
[2] Centre for Artificial Intelligence and Robotics, Hong Kong Institute of Science & Innovation, Chinese Academy of Sciences
{shuxin.li, xinrun.wang, wanqi001, boan}@ntu.edu.sg, youzhi.zhang@cair-cas.org.hk, cerny@disroot.org

Abstract

Pursuit-evasion games on graphs model the coordination of police forces chasing a fleeing felon in real-world urban settings, using the standard framework of imperfect-information extensive-form games (EFGs). In recent years, solving EFGs has been largely dominated by the Policy-Space Response Oracle (PSRO) methods due to their modularity, scalability, and favorable convergence properties. However, even these methods quickly reach their limits when facing large combinatorial strategy spaces of the pursuit-evasion games. To improve their efficiency, we integrate the pre-training and fine-tuning paradigm into the core module of PSRO – the repeated computation of the best response. First, we pre-train the pursuer's policy base model against many different strategies of the evader. Then we proceed with the PSRO loop and fine-tune the pre-trained policy to attain the pursuer's best responses. The empirical evaluation shows that our approach significantly outperforms the baselines in terms of speed and scalability, and can solve even games on street maps of megalopolises with tens of thousands of crossroads – a scale beyond the effective reach of previous methods.

1 Introduction

Preventing, deterring, and detecting crime to ensure the safety and security of the population is an immensely serious task entrusted to various governmental law enforcement forces. Because of its importance, many game-theoretic models have been developed for safeguarding not just the general public, but also critical infrastructure, using only limited resources (Sinha et al. 2018). Among the problems studied, one stands out due to its unflattering statistic: police pursuits probably "injure or kill more innocent bystanders than any other kind of force" (Rivara and Mack 2004). It is hence of extreme importance to devise scalable methods to efficiently coordinate multiple police units to capture a fleeing criminal as soon as possible to minimize casualties and property damages. In the paper, this scenario is commonly referred to as a pursuit-evasion game, and it is modeled via a two-player zero-sum imperfect-information extensive-form game. The extensive-form games provide a versatile framework capable of representing multiple agents, imperfect information, and stochastic events, and they have a delineated solution in the form of Nash equilibrium. Many methods have been formulated for solving this type of game, including linear programs (Shoham and Leyton-Brown 2008), double-oracle algorithms (McMahan, Gordon, and Blum 2003), counterfactual regret minimization (CFR) (Zinkevich et al. 2008), or policy-space response oracles (PSRO) (Lanctot et al. 2017). These algorithms are domain-agnostic and were successfully applied to security problems as well. For example, the double-oracle algorithms constitute the cores of several solvers of attacker-defender scenarios deployed in the real world (Jain et al. 2011), and there exist even CFR or PSRO variants able to solve shallow pursuit-evasion games with many pursuers (Li et al. 2021), or deep pursuit-evasion games with a smaller space of legal actions (Xue et al. 2021; Xue, An, and Yeo 2022).

The contemporary state-of-the-art algorithms for solving extensive-form games may be roughly divided into two groups: no-regret methods derived from CFR, and incremental strategy-space generation methods of the PSRO framework. Both groups have their advantages and disadvantages. The no-regret methods have strong convergence guarantees, but they run out of memory soon or fail to converge when solving games with deep game trees since computing the regret values therein is excessively time-consuming. Because large-scale pursuit-evasion games inevitably involve very long time steps and have extremely large combinatorial action spaces due to the need to coordinate multiple law enforcement units[1], we focus on the second group, the PSRO methods. PSRO is a generalization of the double-oracle algorithm, embedded with reinforcement learning (RL) to handle large strategy spaces. In case the learning is sufficiently precise, the PSRO methods inherit the guarantees of convergence to Nash equilibrium of the double-oracle. PSRO works in iterations, computing best responses to equilibria of meta-games gradually increasing in size. Finding the best responses constitutes the PSRO's key bottleneck, and several recent adaptations attempted to mitigate it through parallelization (Balduzzi et al. 2019; McAleer et al. 2020). However, these methods still compute the responses from scratch, restarting the computation in each iteration,

*Corresponding Author

[1] For example, we solve games on a graph with degree 13 and 6 law enforcement units. The pursuer may hence choose from up to $13^6 \approx 4.8 \cdot 10^6$ actions in each single information set.

which is the main obstacle of scaling the algorithms to the real-world-sized problems. In this work, we show that by pre-training the best responses, and later fine-tuning them, we can solve pursuit-evasion games which are both deep and have a large branching factor.

Pre-training and fine-tuning is a celebrated approach from the area of natural language processing (NLP). For example, BERT (Devlin et al. 2019) is a pre-training method that can learn a general-purpose "language-understanding" model on a large text corpus and then use this model for downstream NLP tasks through fine-tuning it. Pre-training is also a key component of the GPT model (Radford et al. 2018), and was used in AlphaGo (Silver et al. 2016) and AlphaStar (Vinyals et al. 2019) to imitate the behaviors in high-quality human behavioral data. Despite the huge impact of the pre-training/fine-tuning approach on NLP and partially also on computer vision (Szegedy et al. 2015; He et al. 2016), it is still rarely used outside these areas (Han et al. 2021). Especially in the context of general game theory when human behavioral data is not available, the question of whether this approach can be used to improve the efficiency of equilibrium-finding methods remains unanswered.

To fill this gap, we propose a two-stage method for solving large-scale pursuit-evasion games by integrating the pre-training and fine-tuning paradigm into the PSRO framework. Our contributions are threefold: i) we modify the graph embedding algorithm LINE to learn a better game state representation that can retain both the structure information of the graph and the node information related to the exit nodes; ii) we show how to pre-train a pursuer's policy base model against vastly different strategies of the evader with multi-task reinforcement learning; and iii) we formulate a novel best-response oracle in PSRO that fine-tunes the pre-trained pursuer's policy. We perform extensive empirical analysis in large-scale pursuit-evasion games with both synthetic and real-world graphs to show that our two-stage algorithm outperforms other baselines in both solution quality and scalability, which demonstrates the effectiveness of pre-training and fine-tuning. To the best of our knowledge, we are the first to introduce the pre-training and fine-tuning paradigm into an equilibrium-finding algorithm without using human behavioral data.

2 Background and Related Work

2.1 Two-Player Extensive-Form Games

A two-player zero-sum imperfect-information extensive-form game (EFG) (Shoham and Leyton-Brown 2008) can be represented as a tuple $(N, H, A, P, \mathcal{I}, u)$. Here, $N = \{1, 2\}$ is a set of players, and H is a set of game's histories. We use the empty sequence $\emptyset$ to represent the root node (i.e., the start point) of the game tree, included in H. Note that every prefix of a sequence in H is also in H. Z refers to a set of terminal histories, and it is a subset of H. A is a set of available actions for every non-terminal history. More specifically, $A(h) = \{a : (h, a) \in H\}$ refers to a set of available actions at a non-terminal history $h \in H$. P is a player function identifying a player who takes an action at history h. $\mathcal{I}$ denotes a set of information sets of both players, such that $\mathcal{I}_i$ forms a partition over histories h where player i takes action. It means that player i cannot distinguish between the histories in the same information set and $P(h_1) = P(h_2)$ and $A(h_1) = A(h_2)$ for any $h_1, h_2 \in I_i$. u represents a utility function, a mapping from the set of terminal histories to real numbers, i.e., $u : Z \to \mathbb{R}$. We consider only zero-sum games, hence $u_1(z) = -u_2(z), \forall z \in Z$.

A behavior strategy σ_i of player i assigns one probability distribution over $A(I_i)$ for every information set of the player. A strategy profile σ is then a tuple of one strategy for each player, i.e., $\sigma = (\sigma_1, \sigma_2)$. When both players play according to σ, we may compute the reaching probability for every history h, and we denote it as $\pi^{\sigma}(h)$. The expected utility of player i given a strategy profile σ is $u_i(\sigma) = \sum_{h \in Z} \pi^{\sigma}(h) u_i(h)$. We consider the canonical solution concept Nash equilibrium (NE) (Nash 1950), which is a strategy profile such that no player can increase their utility by unilaterally deviating. Formally, a strategy profile σ is an NE if it satisfies $\forall i \in N, u_i(\sigma) \geq \max_{\sigma'_i \in \sum_i} u_i(\sigma'_i, \sigma_{-i})$, where σ_{-i} refers to all the strategies in σ except σ_i. Given $\epsilon > 0$, a strategy profile σ is said to be an ϵ-Nash equilibrium if it is not possible for any player to gain more than ϵ in expected payoff by unilaterally deviating from their strategy, i.e., $\forall i \in N, \max_{\sigma'_i \in \sum_i} u_i(\sigma'_i, \sigma_{-i}) - u_i(\sigma) \leq \epsilon$.

2.2 Policy-Space Response Oracles

Double oracle is an incremental strategy-space generating algorithm that is guaranteed to converge to Nash equilibrium in two-player zero-sum games. The idea of the algorithm is to gradually expand a small meta-game consisting of only a subset of all possible actions of the players by including their best responses (from the entire action space) against the equilibrium of the meta-game. Once all the best responses are already present in the meta-game, which means that the current solution cannot be improved, the algorithm terminates. The PSRO is a generalization of the double oracle, but instead of actions, PSRO uses policies as choices in the meta-game. Due to its favorable scalability, PSRO achieved state-of-the-art performance in large-scale two-player zero-sum games (Lanctot et al. 2017). In PSRO, every player starts with a random policy, and this profile constitutes the initial meta-game. Then it proceeds as in double-oracle: it computes the best response policies of all players and adds them to the meta-game. For computing the best responses we may employ any RL algorithm. The meta-game in PSRO is represented as an empirical game matrix, generated by having each policy of the first player interact with each policy of the second player and recording the average utilities as entries. The solution (i.e., the meta-strategy) is computed from the empirical game matrix using any meta-solver. Once PSRO converges, it outputs the final distribution over population policies which approximates an NE of the original game. Neural Fictious Self-Player (NFSP) (Heinrich and Silver 2016) can be seen as a special case of PSRO with uniform distributions as meta-strategies.

A notable disadvantage of PSRO is the fact that training the best response reinforcement-learning policy is computationally demanding, making it too slow for large games. Several recent works were dedicated to ameliorating this issue. For example, McAleer et al. introduced the Pipeline PSRO (P2SRO) that parallelizes the computation while maintaining the convergence guarantees through a hierarchical pipeline of reinforcement-learning workers (McAleer et al. 2020). Through various speedup techniques, contemporary PSRO-based methods mastered some large games like Stratego (in the case of P2SRO) or Starcraft (Vinyals et al. 2019). Yet, even these breakthroughs still require computing the best response from scratch in each iteration.

2.3 Multi-Task Reinforcement Learning

Multi-task learning is an area of machine learning largely applied to problems in natural language processing or computer vision due to its ability to significantly improve the learning efficiency and prediction accuracy through joint learning (Zhang and Yang 2017). Its subfield applied to reinforcement-learning (RL) tasks have recently drawn the attention of the RL community as well, motivated by the idea that similar decision-making policies may be applied to different tasks in similar environments (Ruder 2017).

Each RL task is modeled as a Markov decision process (MDP), and the goal of RL is to choose actions maximizing the cumulative reward. Wilson et al. suggest that to model a distribution over MDPs, we may employ a hierarchical Bayesian infinite mixture model (Wilson et al. 2007). In this approach, the previously learned distributions are used as an informative prior for model-based Bayesian RL when solving a new MDP. Multiple tasks may be trained simultaneously using a framework of shared experiences (Vuong et al. 2019). This framework uses task-specific rewards to identify similar parts defined as shared regions which can guide the experience-sharing of task policies. Recent work has shown that multi-task pre-training with fine-tuning on new tasks performs equally or better than meta-learning pre-training with meta adaptation in RL tasks (Mandi, Abbeel, and James 2022). This motivates the adoption of multi-task learning as the pre-training algorithm in this work.

3 Problem Statement

Now we move to introduce the class of the game we are interested in: the pursuit-evasion games. A pursuit-evasion game is an imperfect-information EFG with two players – the pursuer and the evader, hence $N = \{\mathbf{p}, e\}$. We assume the pursuer has control over n law enforcement units, i.e., $\mathbf{p} = (p_1, p_2, ..., p_n)$, equipped with tracking devices capable of monitoring the real-time location of the evader[2]. Because in reality both players act at the same time, a pursuit-evasion game is a simultaneous-move EFG in which the evader moves first, followed by the pursuer who makes decisions without being aware of the current action of the evader.

Every game is played on an urban road map, which can be represented as a graph $G = (V, E)$, where V is a vertex set, and E is an edge set. A subset $\overline{E} \subset E$ denotes the set of exit nodes from which the evader can escape. For a vertex $v \in V$, let $\mathcal{N}(v)$ define the set of neighborhood vertices of v. Let $l_0^e, l_0^{\mathbf{p}}$ be the initial locations of evader and pursuer. Note that $l_0^{\mathbf{p}} = (l_0^{p_1}, l_0^{p_2}, ..., l_0^{p_n})$. By T we denote the number of steps in which the game terminates. At time step $t < T$, the history of the game is a sequence of past locations of both players, $h = (l_0^e, l_0^{\mathbf{p}}, ..., l_{t-1}^e, l_{t-1}^{\mathbf{p}})$. Because the evader cannot know the pursuer's location, evader's information set can be defined as $I_e = \{h | h = (l_0^e, l_0^{\mathbf{p}}, l_1^e, *, ..., l_{t-1}^e, *)\}$, where $*$ is any possible location of pursuer. The actions available to the evader are the neighboring vertices of the evader's current location, i.e., $A_e(h) = \mathcal{N}(l_{t-1}^e)$. The evader makes their decision $l_t^e \in A_e(h)$ based on the information available in set I_e. When the evader moves from l_{t-1}^e to l_t^e, the history becomes $h = (l_0^e, l_0^{\mathbf{p}}, ..., l_{t-1}^e, l_{t-1}^{\mathbf{p}}, l_t^e)$. Then it is up to the pursuer to act in the information set defined as $I_{\mathbf{p}} = \{h | h = (l_0^e, l_0^{\mathbf{p}}, ..., l_{t-1}^e, l_{t-1}^{\mathbf{p}}, *)\}$. The action set of the pursuer at the current history h is a Cartesian product of the sets of neighboring vertices of each unit, i.e., $A_{\mathbf{p}}(h) = \{(l^{p_1}, l^{p_2}, ..., l^{p_n}) | l^i \in \mathcal{N}(l_{t-1}^i), \forall i \in \{p_1, p_2, ..., p_n\}\}$. When the pursuer moves from $l_{t-1}^{\mathbf{p}}$ to $l_t^{\mathbf{p}}$ after choosing an action from $A_{\mathbf{p}}(h)$, it is up to the evader to act again. This process repeats until a termination condition is met. The game ends (i) when the pursuer catches the evader, i.e., $l_t^e \in l_t^{\mathbf{p}}$; (ii) when the evader escapes; or (iii) when the game reaches the time step T. In cases (i) and (iii) the pursuer obtains a single reward (1), and the evader gets a single penalty (-1). Otherwise, if the evader successfully escapes to the outside world, the pursuer obtains a single penalty (-1), and the evader gets a single reward (1).

Because a pursuit-evasion game is an EFG, we may leverage the standard methods for finding its NE. However, applying these methods directly is not possible because of the large combinatorial action space of pursuit-evasion games. Some adaptations hence consider various (and often domain-specific) incremental generation strategies (Bosansky et al. 2014, 2015; Zhang et al. 2019), or introduce new strategy representations in attempt to avoid the curse of dimensionality associated with the game's strategy space (Li et al. 2021). However, neither of these approaches scale to deep games with high number of time steps. In order to solve such games, in this work, we develop a PSRO-based algorithm embedded with pre-training and fine-tuning of pursuer's strategies to accommodate both the high branching factor and the long interactions between the players.

4 Pre-Training and Fine-Tuning in PSRO

In this section, we describe our algorithm for solving large pursuit-evasion games. The algorithm may be divided into two stages, as shown in Figure 1: the pre-training stage and the fine-tuning stage. In the pre-training stage, we first learn a good representation for the state and then train the policy base model of the pursuer against different evader strategies using multi-task learning. In the fine-tuning stage, we use the learned base model in the pursuer's best response oracle in PSRO to arrive at the game's equilibrium.

4.1 Learning the Game State Representation

First, we consider the computation of the pursuer's best response during the first stage. As described in the previous

[2] This version of pursuit-evasion games is also called the NEtwork purSuiT games (NEST) (Zhang et al. 2019) or the Network Security Games (NSGs) (Xue et al. 2021; Xue, An, and Yeo 2022)

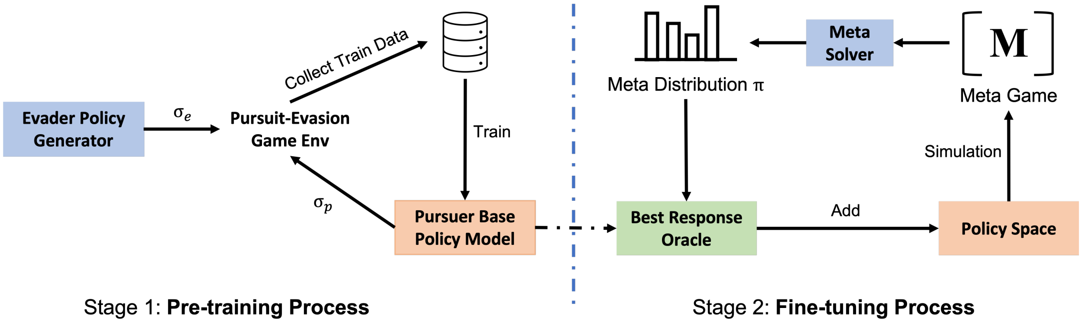


Figure 1: Structure of our PSRO-based method embedded with the pre-training and fine-tuning stages.

section, the information set of the pursuer is determined by the past locations of both players, $(l_0^e, l_0^{\mathbf{p}}, ..., l_{t-1}^e, l_{t-1}^{\mathbf{p}})$. This history may be considered as the state in the best response computation task. If we use only the vertex index in the graph to represent the state, this representation would not retain the graph's structural information as the index is basically meaningless. To obtain a better state representation used for learning, we adopt a graph embedding algorithm to learn the representation of the graph vertex and replace the vertex index with the vertex's representation. The embeddings need to preserve not only the structure information of the graph but also the node information proximity. We modify an existing graph embedding algorithm to achieve it.

Large-scale information network embedding (LINE) is a graph embedding algorithm which aims to embed large information networks into low-dimensional vector spaces (Tang et al. 2015). This method can preserve both the first-order and second-order proximities. The first-order proximity refers to the local pairwise proximity between the vertices. The second-order proximity assumes that vertices sharing many connections to other vertices are similar.

Note that in pursuit-evasion games, the exit nodes are important for both players. We hence use the information related to the exit nodes as a "node information", and we require the nodes with similar node information to have similar embeddings, whether they are connected or not. Therefore, except for the above two proximities, we also intend the preserve this so-called node information proximity.

Node Information. Formally, we define the node information of a vertex as the shortest distances between the vertex and the exit nodes. We use D_{NI}^i to denote the node information vector for vertex $v_i \in V$.

Node Information Proximity. For any pair of vertices (v_i, v_j), we use the cosine similarity as a node information metric, defined as follows:

$$f_{NI}(D_{NI}^i, D_{NI}^j) = \frac{D_{NI}^i \cdot D_{NI}^j}{||D_{NI}^i|| \times ||D_{NI}^j||}. \tag{1}$$

Objective Function. For any pair of vertices (v_i, v_j), we define the node similarity between v_i and v_j as a joint probability between the two vertices:

$$p_{NI}(v_i, v_j) = \frac{1}{1 + \exp(-u_i^T \cdot u_j)}, \tag{2}$$

where $u_i \in R^d$ is the low-dimensional vector representation of vertex v_i. Moreover, we denote the empirical probability of the above distribution $p_{NI}(\cdot,\cdot)$ over the space $V \times V$ as $\hat{p}_{NI}(i,j) = f_{NI}(D_{NI}^i, D_{NI}^j)$, where f_{NI} is the cosine similarity function. Therefore, to preserve the node information proximity, we minimize the following objective function:

$$O_d = d(\hat{p}_{NI}(\cdot,\cdot), p_{NI}(\cdot,\cdot)), \tag{3}$$

where $d(\cdot,\cdot)$ is a distance between the two distributions. In case of the KL-divergence, the function reduces to:

$$O_{KL} = \sum_{\forall(v_i,v_j)} f_{NI}(D_{NI}^i, D_{NI}^j) \log \frac{f_{NI}(D_{NI}^i, D_{NI}^j)}{p_{NI}(v_i, v_j)}.$$

After eliminating the constants, we finally arrive at:

$$O_{KL} = -\sum_{\forall(v_i,v_j)} f_{NI}(D_{NI}^i, D_{NI}^j) \log p_{NI}(v_i, v_j). \tag{4}$$

To preserve the structure information of the graph and node information at the same time, we add O_{KL} to the objective function of the LINE algorithm.

State Representation. After obtaining the embedding for all vertices in the graph, we can replace the vertex index with its corresponding embedding in the information set representation $(l_0^e, l_0^{\mathbf{p}}, ..., l_{t-1}^e, l_{t-1}^{\mathbf{p}})$. However, the size of the state representation quickly increases with time step. For this reason, we represent a state using only the locations of both players and the number of remaining time steps. Note that such simplification still carries the entire information as the locations of both players in the next time step depend only on their current positions and their actions. In the experimental section, we show that this new state representation performs equally or better than the traditional state representation using historical locations.

4.2 Learning the Pursuer's Policy Base Model

The computation of the pursuer's best response strategy against a fixed evader's strategy can be regarded as an RL task. The environment of this task depends mainly on the game's rules and the fixed evader's policy. More specifically, the task changes in accordance with the strategy of the evader. In PSRO, the pursuer's best response strategy is computed in each iteration, each characterizing a specific

Algorithm 1: Pre-train Policy Model

```
 1: Initialize a policy network $\theta$ and policy buffer *B*;
 2: **for** train epoch in $\{1, 2, ..., n\}$ **do**
 3:     **for** number of tasks sampled **do**
 4:         Generate an evader's strategy $\sigma_e$ randomly;
 5:         Sample training data using $\sigma_e$ and $\theta$;
 6:         Add sampled training data into buffer *B*;
 7:     **end for**
 8:     Train policy $\theta$ using training data in *B*;
 9:     Clear buffer *B*;
10: **end for**
11: **return** policy network $\theta$.
```

RL task identified by a different strategy of the evader. Our assumption here is that these RL tasks may be seen as similar, following the same distribution, and determined by the evader's strategy space.

We use this similarity in practice to speed up the computation of the pursuer's best response in PSRO. To this end, we pre-train the policy base model of the pursuer and quickly fine-tune it to adapt to a new, similar RL task. In order to do so, we need to solve two problems: how to generate the tasks, and how to learn the policy base model.

Task Generation. Because the RL tasks are determined by the evader's strategy, to obtain enough tasks for training the policy base model of the pursuer, we need to generate as many different evader strategies as possible. Recall that the goal of the evader is to escape from exit nodes in the graph. We hence focus on so-called High-Level Actions of the evader, which proved useful in earlier work (Xue et al. 2021; Xue, An, and Yeo 2022). In this action representation, the evader chooses which exit node to escape from and then samples one path from the initial location to this exit node, rather than deciding where to go in the next step as in regular strategies. In the PSRO algorithm, when computing the best response strategy for the pursuer, the fixed evader strategy is a mixed strategy produced by the meta distribution and previous best response strategies. Without loss of generality, we generate the mixed strategy for the evader randomly. In practice, it means that if there are n exit nodes, we randomly generate a distribution σ_e over n exit nodes. This way we can obtain enough RL tasks for pre-training across a wide range of meaningful evader's strategies.

Policy Learning. After obtaining enough RL tasks, we move to the pre-training phase of the pursuer's policy base model. Our priority is that the model generalizes well, in other words, it needs to be able to fine-tune quickly to deliver acceptable performance when faced with a new task determined by a previously unseen evader's strategy. For this purpose, we adopt a multi-task reinforcement learning approach to guide the pre-training process. By doing so, we complete all the training tasks simultaneously with a single RL policy network. When adapting to a new task, we fine-tune the policy network using the same algorithm as in training. The framework of multi-task reinforcement learning is agnostic to the underlying base RL algorithm. We opt for a popular on-policy method called proximal policy optimization (PPO) (Schulman et al. 2017), that is known to perform extremely well across a wide range of tasks. The diagram and the entire procedure of pre-training are depicted in Figure 1 and Algorithm 1.

Algorithm 2: PSRO with Pre-trained Policy Model

```
**Require:** pre-trained policy $\theta_p$ of the pursuer
 1: Initialize policy $\theta_e^0$ of the evader, $\Pi_e = \{\theta_e^0\}$;
 2: Initialize policy $\theta_p^0 = \theta_p$ of the pursuer, $\Pi_p = \{\theta_p^0\}$;
 3: **while** epoch $i$ in $\{1, 2, ..., n\}$ **do**
 4:     Compute best response $\theta_e^i$ of the evader against $\sigma_p$;
 5:     $\theta_p^i = \theta_p$;
 6:     **for** few episodes **do**
 7:         Sample $\theta_e \sim \sigma_e$;
 8:         Train oracle $\theta_p^i$ over $\phi \sim (\theta_e, \theta_p^i)$;
 9:     **end for**
10:     $\Pi_e = \Pi_e \cup \{\theta_e^i\}$, $\Pi_p = \Pi_p \cup \{\theta_p^i\}$;
11:     Compute the missing entries in $U^\Pi$ from $\Pi$;
12:     Compute a meta-strategy $\sigma$ from $U^\Pi$;
13: **end while**
**Ensure:** current solution strategy $\sigma$ and $\Pi$.
```

First, we initialize a policy network θ and a buffer B. In the pre-training process, we use the experience-sharing approach to conduct multi-task learning. We begin the training by obtaining some tasks from several generated evader's strategies. Then we use the current policy θ to sample the training data from these tasks and add the data into the buffer B. We train the policy θ based on the data in the buffer using the base RL algorithm (e.g., PPO). We finish the epoch by emptying the buffer. This whole process is repeated for a given number of training epochs.

Since the pursuer controls several law enforcement units, their joint action space grows exponentially in the number of units. As a consequence, training the joint action strategy over the large joint action space using PPO would be highly ineffective. To mitigate this issue, we replace the vanilla PPO with the Multi-Agent PPO (MAPPO) (Yu et al. 2021), which is variant of PPO with centralized value function inputs. The motivation for choosing a multi-agent RL algorithm is that all units need to cooperate to capture the escaping evader. Computing the best response of the pursuer with many units may hence be regarded as a multi-agent cooperative RL problem. To use the MAPPO as the base RL algorithm for multi-task reinforcement learning, we only need to change the training pattern in Line 8 of Algorithm 1. We note that any multi-agent cooperative RL algorithm may be used in the pre-training stage we devised.

4.3 Designing an Oracle using the Base Model

Here, we describe how to integrate the pre-trained model of the pursuer's policy into the PSRO framework. The diagram of the integration and its method are shown in Figure 1 and Algorithm 2. At the beginning, we initialize the evader's strategy randomly and the pursuer's strategy by using the pre-trained policy base model. Then we follow the standard PSRO loop: in each iteration we compute the best responses of both players using their appropriate oracles, we update the meta-game matrix, and compute the meta-strategy. Without a doubt, the key component of PSRO is the computations of best responses, which we derive in the next two paragraphs.

Best Response Oracle for Evader. Since the goal of the evader is to escape through the exit nodes in the graph, and they cannot obtain additional information about the pursuer's location during the game, their best strategy should be to follow a path from their initial location to one exit node. However, evaluating every path to select the best strategy is inefficient because the number of paths grows exponentially with the number of time steps. Therefore, we employ the High-Level Actions we described earlier in the task generation part. The evader only needs to select which exit node to escape from and sample one path from the initial location to the exit node. Then we estimate the value of each exit node through simulations. The best response strategy of the evader is selected according to these values.

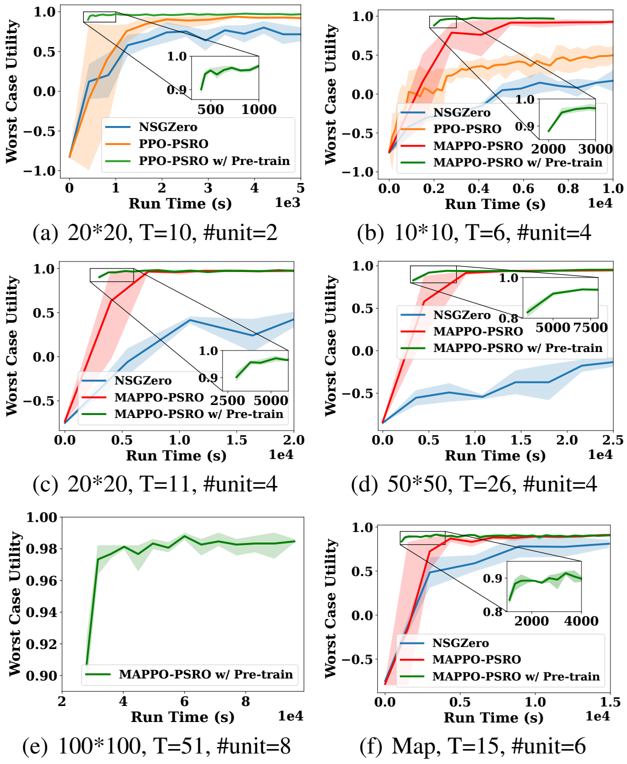
Best Response Oracle for Pursuer. The primary difference between vanilla PSRO and our version of PSRO with the integrated pre-training and fine-tuning lies in the best response computation for the pursuer. Instead of training the best response strategy from scratch, we fine-tune the pre-trained policy base model to arrive at the best response strategy. As shown in Algorithm 2, we start with the pre-trained policy model (Line 5) and sample the training data against the evader's mixed strategy. Then we fine-tune the pre-trained model on the sampled data using the same RL algorithm as in pre-training, but with a smaller learning rate (Lines 6-9).

5 Empirical Evaluation

To evaluate the effectiveness of our two-stage algorithm, we compare it against the vanilla PSRO and NSGZero (Xue, An, and Yeo 2022), which is the state-of-the-art algorithm for solving pursuit-evasion games, in games with different sizes of graphs[3]. Furthermore, we perform several ablation studies to identify the effects of each component of our algorithm we introduced.

5.1 Experimental Setting

Pursuit-evasion games are played on graphs which sizes in part determine the games' sizes. In the experiments, we use different sizes of synthetic graphs and a real-world map. For the synthetic graphs, we generate different sizes of grid worlds, and for the real-world graph, we extract the graph from the Singapore map via OSMnx (Boeing 2017). Since we focus more on the quality of the pursuer's strategy, the measurement we plot is the worst-case utility for the pursuer, which is their utility against the evader's best response strategy. The values of all parameters can be found in the Appendix. To increase the robustness, we run every experiment with three seeds and plot the average values and variances in every figure. Experiments are performed on a workstation with a ten-core 3.3GHz Intel i9-9820X CPU and NVIDIA RTX 2080 Ti GPU.



(a) 20*20, T=10, #unit=2

(b) 10*10, T=6, #unit=4

(c) 20*20, T=11, #unit=4

(d) 50*50, T=26, #unit=4

(e) 100*100, T=51, #unit=8

(f) Map, T=15, #unit=6

Figure 2: Qualitative results on games with different graphs.

5.2 Experimental Results

Synthetic Graphs. First, we use games that play on synthetic graphs as testbeds. Figure 2(a) shows the results of the game on a 20*20 grid graph. Since the number of units is only two in this game, we use PPO as the base RL algorithm of the multi-task reinforcement learning to pre-train the pursuer's policy base model (PPO-PSRO). From the figure, we can find that PPO-PSRO and NSGZero show similar performance, while both perform worse than our algorithm (PPO-PSRO with Pre-train). When performing PSRO to get the NE strategy, although it takes some time in pre-training, our algorithm can start with a high worst-case utility and be improved by fine-tuning, which shows the effectiveness of pre-training and the necessity of fine-tuning.

Next, we consider games with more units controlled by the pursuer. Figure 2(b) shows the results in the game with four units played on a 10*10 grid graph. In this game, we use multi-agent RL algorithm (i.e., MAPPO) as the best response computation strategy to avoid the exponentially growing action space. The results show that MAPPO-PSRO and MAPPO-PSRO with Pre-train perform better than PPO-PSRO and NSGZero, which indicates the large joint action space hampers the performance of PPO. Although NSGZero tries to mitigate the issue produced by large action space by only considering legal actions and taking the legal action as the input, NSGZero still suffers from the issue of combinatorial action space when the graph is compact. Figures 2(b)-2(d) show the results in games with four units played on different sizes of graphs. It shows our algorithm (MAPPO-

[3] As we mentioned in the introduction, CFR variants cannot effectively solve games with such long time horizons, so we do not include them in the experimental comparison.

PSRO with Pre-train) performs best in all these games, although it spends much time on pre-training. To further evaluate the scalability of our algorithm, we test our algorithm on the game played on a 100*100 grid graph, which involves 10000 nodes and 39600 edges. The number of units is set to be eight, and the time horizon is 51. Figure 2(e) shows only the results of our two-stage algorithm (PSRO with Pre-train) since PSRO and NSGZero cannot converge to a satisfactory result in the limited time. It indicates that only our algorithm can solve such large games within an acceptable time.

Real-world Graphs. Finally, we evaluate our algorithm in a real-world graph extracted from the Singapore map, which involves 372 nodes and 1098 edges, and its max degree is 13. Figure 2(f) shows the results which exhibit the our two-stage algorithm can still perform better and faster than the vanilla PSRO. It indicates that our algorithm can perform well in large-scale games regardless of the graph structure.

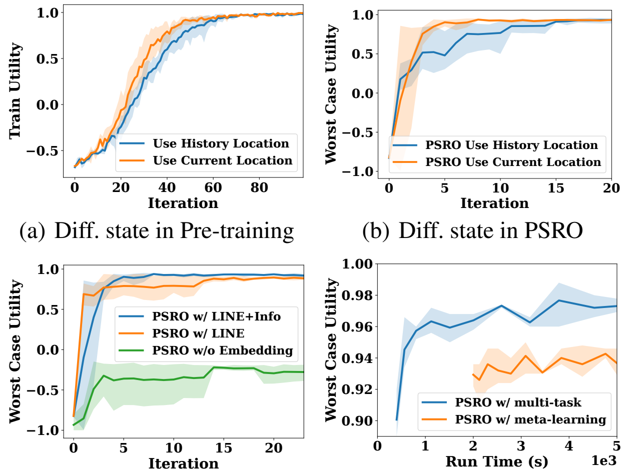
5.3 Ablation Study

To evaluate the effectiveness of each component in our algorithm – especially whether the state representation and the multi-task learning improve the performance – we ran several ablation experiments in a game played on a 20*20 grid graph. The pursuer in this game controls two units and the time horizon is set to be 10. We employ the PPO as the algorithm for computing the best response strategy.

Different State Representation. As described in Section 4.1, instead of using the historical locations of both players, we only use the current positions of both players and the remaining time as the state representation. To compare these two state representations, we perform the pre-training process and PSRO using these two state representations. Both two state representations use the same node embeddings. Figures 3(a)-3(b) show the results. We found that the performances of these two state representations are similar, and the new state representation performs even better. It means that new state representation can perform equally or even better than the state representation using historical locations.

To evaluate the effectiveness of the graph embedding algorithm, we take state representation without graph embedding (only use the vertex index) as the baseline, and the LINE algorithm is also taken as another baseline to test the effect of node information. Figure 3(c) shows the results. It indicates that PSRO with LINE+Information converges to the highest worst-case utility, and PSRO without embedding cannot converge to a high worst-case utility. PSRO with LINE and PSRO with LINE+Information perform much better than PSRO without embedding, which means that the node embeddings can improve the performance of PSRO. Although PSRO with LINE performs better than PSRO with LINE+Information at the beginning, PSRO with LINE+Information seems to converge to a higher worst-case utility and runs more stable than PSRO with LINE. It means that the node information can provide some help for solving games and make the learning phase stable.

Different Pre-training Algorithms. Meta-learning aims to bootstrap from a set of tasks to learn faster on a new task. Therefore, the meta-learning algorithm also can be used to replace the multi-task reinforcement learning algorithm in our framework to pre-train the policy base model of the pursuer. Model-Agnostic Meta-Learning Algorithm (MAML) (Finn, Abbeel, and Levine 2017) is one of the popular meta-learning algorithms. It is compatible with any model trained with gradient descent and applicable to different learning problems, including RL tasks. More details about meta-learning can be found in the Appendix.



(a) Diff. state in Pre-training (b) Diff. state in PSRO

(c) Diff. representation method (d) Diff. pre-training algorithm

Figure 3: Results of the ablation studies.

To test the performance of our algorithm when using different pre-training algorithms, we pre-train the policy base models using multi-task learning and meta-learning algorithms, respectively. Then we fine-tune these two pre-trained models under the PSRO framework. Figure 3(d) shows the results. It demonstrates that both methods can start with a high worst-case utility which means both pre-trained policy models are well trained. From the figure, we can also find that the meta-learning algorithm needs more time to pre-train the policy base model than multi-task learning. Furthermore, the pre-trained model using multi-task learning seems to be better fine-tuned than the pre-trained model using meta-learning. It indicated that multi-task learning tends to be computationally cheaper than meta-learning though both algorithms can get good pre-trained models.

6 Conclusion

In this paper, we propose a two-stage method for solving large-scale pursuit-evasion games by introducing the pre-training and fine-tuning paradigm under the PSRO framework. Instead of training the pursuer's best response strategy from scratch, we fine-tune a pre-trained policy model to speed up the computation of the best response in the PSRO algorithm. To get a pre-trained policy model, we employ the graph embedding algorithm to get a good state representation and adopt the multi-task reinforcement learning algorithm to train the policy base model. Finally, extensive experimental results in games on both synthetic and real-world graphs show that our two-stage algorithm outperforms other baselines in terms of speed and scalability.

Acknowledgments

This research is supported by the National Research Foundation, Singapore under its Industry Alignment Fund – Pre-positioning (IAF-PP) Funding Initiative, the InnoHK Fund, and the SIMTech-NTU Joint Laboratory on Complex Systems. 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 National Research Foundation, Singapore.

References

Balduzzi, D.; Garnelo, M.; Bachrach, Y.; Czarnecki, W.; Perolat, J.; Jaderberg, M.; and Graepel, T. 2019. Open-ended learning in symmetric zero-sum games. In *ICML*, 434–443.

Boeing, G. 2017. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65: 126–139.

Bosansky, B.; Jiang, A. X.; Tambe, M.; and Kiekintveld, C. 2015. Combining compact representation and incremental generation in large games with sequential strategies. In *AAAI*, 812–818.

Bosansky, B.; Kiekintveld, C.; Lisy, V.; and Pechoucek, M. 2014. An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information. *Journal of Artificial Intelligence Research*, 51: 829–866.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 4171–4186.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 1126–1135.

Han, X.; Zhang, Z.; Ding, N.; Gu, Y.; Liu, X.; Huo, Y.; Qiu, J.; Yao, Y.; Zhang, A.; Zhang, L.; et al. 2021. Pre-trained models: Past, present and future. *AI Open*, 2: 225–250.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*, 770–778.

Heinrich, J.; and Silver, D. 2016. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*.

Jain, M.; Korzhyk, D.; Vaněk, O.; Conitzer, V.; Pěchouček, M.; and Tambe, M. 2011. A double oracle algorithm for zero-sum security games on graphs. In *AAMAS*, 327–334.

Lanctot, M.; Zambaldi, V.; Gruslys, A.; Lazaridou, A.; Tuyls, K.; Pérolat, J.; Silver, D.; and Graepel, T. 2017. A unified game-theoretic approach to multiagent reinforcement learning. In *NeurIPS*, 4193–4206.

Li, S.; Zhang, Y.; Wang, X.; Xue, W.; and An, B. 2021. CFR-MIX: Solving imperfect information extensive-form games with combinatorial action space. In *IJCAI*, 3663–3669.

Mandi, Z.; Abbeel, P.; and James, S. 2022. On the effectiveness of fine-tuning versus meta-reinforcement learning. *arXiv preprint arXiv:2206.03271*.

McAleer, S.; Lanier, J.; Fox, R.; and Baldi, P. 2020. Pipeline PSRO: a scalable approach for finding approximate nash equilibria in large games. In *NeurIPS*, 20238–20248.

McMahan, H. B.; Gordon, G. J.; and Blum, A. 2003. Planning in the presence of cost functions controlled by an adversary. In *ICML*, 536–543.

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

Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I.; et al. 2018. Improving language understanding by generative pre-training. https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf. Accessed: 2018.

Rivara, F. P.; and Mack, C. D. 2004. Motor vehicle crash deaths related to police pursuits in the United States. *Injury Prevention*, 10(2): 93–95.

Ruder, S. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shoham, Y.; and Leyton-Brown, K. 2008. *Multiagent Systems: Algorithmic, Game-theoretic, and Logical Foundations*. Cambridge University Press.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587): 484–489.

Sinha, A.; Fang, F.; An, B.; Kiekintveld, C.; and Tambe, M. 2018. Stackelberg security games: Looking beyond a decade of success. In *IJCAI*, 5494–5501.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *CVPR*, 1–9.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. LINE: Large-scale information network embedding. In *WWW*, 1067–1077.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782): 350–354.

Vuong, T.-L.; Nguyen, D.-V.; Nguyen, T.-L.; Bui, C.-M.; Kieu, H.-D.; Ta, V.-C.; Tran, Q.-L.; and Le, T.-H. 2019. Sharing experience in multitask reinforcement learning. In *IJCAI*, 3642–3648.

Wilson, A.; Fern, A.; Ray, S.; and Tadepalli, P. 2007. Multi-task reinforcement learning: A hierarchical Bayesian approach. In *ICML*, 1015–1022.

Xue, W.; An, B.; and Yeo, C. K. 2022. NSGZero: Efficiently learning non-exploitable policy in large-scale network security games with neural monte carlo tree search. In *AAAI*, 4646–4653.

Xue, W.; Zhang, Y.; Li, S.; Wang, X.; An, B.; and Yeo, C. K. 2021. Solving large-scale extensive-form network security games via neural fictitious self-play. In *IJCAI*, 3713–3720.

Yu, C.; Velu, A.; Vinitsky, E.; Wang, Y.; Bayen, A.; and Wu, Y. 2021. The surprising effectiveness of PPO in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*.

Zhang, Y.; Guo, Q.; An, B.; Tran-Thanh, L.; and Jennings, N. R. 2019. Optimal interdiction of urban criminals with the aid of real-time information. In *AAAI*, 1262–1269.

Zhang, Y.; and Yang, Q. 2017. A survey on multi-Task learning. *arXiv preprint arXiv:1707.08114*.

Zinkevich, M.; Johanson, M.; Bowling, M.; and Piccione, C. 2008. Regret minimization in games with incomplete information. In *NeurIPS*, 1729–1736.