Grasper: A Generalist Pursuer for Pursuit-Evasion Problems

Pengdeng Li*
Nanyang Technological University
Singapore
pengdeng.li@ntu.edu.sg

Shuxin Li*
Nanyang Technological University
Singapore
shuxin.li@ntu.edu.sg

Xinrun Wang†
Nanyang Technological University
Singapore
xinrun.wang@ntu.edu.sg

Jakub Černý
Columbia University
New York City, United States
cerny@disroot.org

Youzhi Zhang
CAIR, HKISI, CAS
Hong Kong, China
youzhi.zhang@cair-cas.org.hk

Stephen McAleer
Carnegie Mellon University
Pittsburgh, United States
mcaleer.stephen@gmail.com

Hau Chan
University of Nebraska-Lincoln
Lincoln, Nebraska, United States
hchan3@unl.edu

Bo An
Nanyang Technological University
Singapore
boan@ntu.edu.sg

ABSTRACT

Pursuit-evasion games (PEGs) model interactions between a team of pursuers and an evader in graph-based environments such as urban street networks. Recent advancements have demonstrated the effectiveness of the pre-training and fine-tuning paradigm in Policy-Space Response Oracles (PSRO) to improve scalability in solving large-scale PEGs. However, these methods primarily focus on specific PEGs with fixed initial conditions that may vary substantially in real-world scenarios, which significantly hinders the applicability of the traditional methods. To address this issue, we introduce Grasper, a GeneRAlist purSuer for Pursuit-Evasion pRoblems, capable of efficiently generating pursuer policies tailored to specific PEGs. Our contributions are threefold: First, we present a novel architecture that offers high-quality solutions for diverse PEGs, comprising critical components such as (i) a graph neural network (GNN) to encode PEGs into hidden vectors, and (ii) a hypernetwork to generate pursuer policies based on these hidden vectors. As a second contribution, we develop an efficient three-stage training method involving (i) a pre-pretraining stage for learning robust PEG representations through self-supervised graph learning techniques like graph masked auto-encoder (GraphMAE), (ii) a pre-training stage utilizing heuristic-guided multi-task pre-training (HMP) where heuristic-derived reference policies (e.g., through Dijkstra's algorithm) regularize pursuer policies, and (iii) a fine-tuning stage that employs PSRO to generate pursuer policies on designated PEGs. Finally, we perform extensive experiments on synthetic and real-world maps, showcasing Grasper's significant superiority over baselines in terms of solution quality and generalizability. We demonstrate that Grasper provides a versatile approach for solving pursuit-evasion problems across a broad range of scenarios, enabling practical deployment in real-world situations.

*Equal contribution.
†Corresponding author.

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand.

KEYWORDS

Multi-Agent Learning, Pursuit-Evasion Problems, Generalizability, Pre-training and Fine-tuning, Hypernetwork

ACM Reference Format:
Pengdeng Li*, Shuxin Li*, Xinrun Wang†, Jakub Černý, Youzhi Zhang, Stephen McAleer, Hau Chan, and Bo An. 2024. Grasper: A Generalist Pursuer for Pursuit-Evasion Problems. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 9 pages.

1 INTRODUCTION

The deployment of security resources to detect, deter, and catch criminals is a critical task in urban security [30, 32]. Statistics show that police pursuits probably "injure or kill more innocent bystanders than any other kind of force" [26]. Therefore, it is crucial to come up with scalable approaches for effectively coordinating various security resources, ensuring the swift apprehension of a fleeing criminal to minimize harm and property damage. Due to the adversarial nature between attackers and defenders, game-theoretic models have been used to model various real-world urban security scenarios. In particular, the pursuit-evasion game (PEG) has been extensively employed to model the interactions between a team of pursuers (e.g., police forces) and an evader (e.g., a criminal) on graphs (e.g., urban street networks) [21, 38, 42, 43]. To effectively solve PEGs under various settings, several methods, such as counterfactual regret minimization (CFR) [47] and policy-space response oracles (PSRO) [18], have been developed in the literature. Among these algorithms, PSRO, a deep reinforcement learning algorithm, provides a versatile framework for learning the (approximate) Nash equilibria (NEs) of PEGs (refer to Section 3.2 for an introduction of the PSRO framework). Furthermore, recent works have also integrated the pre-training and fine-tuning paradigm into the PSRO framework to further enhance its scalability [20].

Although many existing works have achieved significant success, they only focus on solving specific PEGs with predetermined initial conditions, e.g., the initial locations of all players and exits, and the time horizon of the game. Unfortunately, these conditions may vary substantially in real-world scenarios, where crimes can occur at

any location in a city and at any time. When the initial conditions change, existing algorithms must solve the new PEG from scratch, which is computationally demanding and time-consuming [20], restricting the real-world deployment of current algorithms. Thus, there is an urgent necessity to develop a new approach capable of solving different PEGs with varying initial conditions effectively.

To this end, we introduce Grasper: a GeneRAlist purSuer for Pursuit-Evasion pRoblems, which can effectively solve different PEGs by generating the pursuer's policies conditional on the initial conditions of the PEGs. Grasper consists of two critical components. First, as the PEG is played on a graph, it is natural to use a graph neural network (GNN) to encode the PEG with the given initial conditions into a hidden vector. Then, inspired by recent work on generalization over games with different population sizes [19], we introduce a hypernetwork to generate the base policy for the pursuer conditional on the hidden vector obtained by the GNN. This generated base policy then serves as a starting point for the pursuer's best response policy training at each PSRO iteration.

Provided the architecture of Grasper, the next problem is how to efficiently train the networks. Multi-task training is one of the common methods to endow networks with generalizability [44]. However, we find that naively applying multi-task training to train the networks of Grasper is inefficient. Furthermore, jointly training the GNN and hypernetwork could be time-consuming as the GNN is only used to encode the initial conditions which are fixed during the game playing. To address these challenges, we propose an efficient three-stage training method. First, we introduce a pre-pretraining stage to train the GNN by using self-supervised graph learning methods such as graph masked auto-encoder (GraphMAE) [15]. Second, we fix the GNN and pre-train the hypernetwork by using a multi-task training procedure where the training data is sampled from different PEGs with different initial conditions. In this stage, to overcome the low exploration efficiency due to the pursuers' random exploration and the evader's rationality, we propose a heuristic-guided multi-task pre-training (HMP) where a reference policy derived by heuristic methods such as Dijkstra is used to regularize the pursuer policy. Finally, we follow the PSRO procedure and obtain the pursuer's best response policy at each iteration by fine-tuning the base policy generated by the hypernetwork.

In summary, we provide three contributions. First, we propose Grasper which is the first generalizable framework capable of efficiently providing highly qualified solutions for different PEGs with different initial conditions. Second, to efficiently train the networks of Grasper, we propose a three-stage training method: (i) a pre-pretraining stage to train the GNN through GraphMAE, (ii) a pre-training stage to train the hypernetwork through heuristic-guided multi-task pre-training (HMP), and (iii) a fine-tuning stage to obtain the pursuer's best response policy at each PSRO iteration. Finally, we perform extensive experiments, and the results demonstrate the superiority of Grasper over different baselines.

2 RELATED WORK

Pursuit-evasion games (PEGs) have been extensively applied to model various real-world problems such as security and robotics [3, 14, 17, 22, 23, 33, 35]. To efficiently solve PEGs and different variants, many algorithms such as value iteration [14] and incremental strategy generation [42, 43] have been introduced. Nonetheless, these methods encounter scalability issues as they typically rely on linear programming. On the other hand, PEG can be viewed as a particular type of two-player imperfect-information extensive-form game (IIEFG). Thus, the algorithms used for solving large-scale IIEFGs, such as counterfactual regret minimization (CFR) [47] and Policy-Space Response Oracles (PSRO) [18], have been applied to tackle large-scale PEGs [21, 38]. However, when solving large-scale PEGs using PSRO, there exist significant computational challenges as it involves computing the best response strategy multiple times. To mitigate this issue, recent research [20] integrates the pre-training and fine-tuning paradigm into PSRO to improve its scalability. Despite the success, all these algorithms are tailored to solve a specific PEG with predetermined initial conditions. When these conditions change, they must recompute the NE strategy from scratch (one to two hours for a PEG on a 10×10 grid map [20]), which hinders their real-world applicability[1]. To address this limitation, we propose Grasper, which uses PSRO to compute the NE strategy and can generate different pursuers' strategies for different PEGs based on their initial conditions without recomputing the NE strategy from scratch.

The generalizability of algorithms and models over different games has gained increasing attention and remarkable progress has been achieved in recent research. Neural equilibrium approximators that directly predict the equilibrium strategy from game payoffs in normal-form games (NFGs) have been theoretically proven PAC learnable [7, 8] and are able to generalize to different games with desirable solution quality [7–9, 24]. However, it remains underexplored when going beyond NFGs. In this work, we make the first attempt to consider the generalization problem in the domain of PEGs, a type of game that has a wide range of real-world applications [14, 22] and is far more complicated than NFGs. We propose a novel algorithmic framework that is able to efficiently solve different PEGs with varying initial conditions and demonstrate the generalization ability through extensive experiments.

Our work is also related to self-supervised graph learning and multi-task learning. Recent works have shown that generative self-supervised learning [13] can be applied to graph learning and outperform contrastive methods which require complex training strategies [25, 31], high-quality data augmentation [41], and negative samples that are often challenging to construct from graphs [46]. Therefore, we employ the recent state-of-the-art, GraphMAE [15], to learn a good representation of a PEG with the given initial conditions. Multi-task learning [27, 44] has been applied to various domains including natural language processing [6], computer vision [11], and reinforcement learning (multi-task RL) [34, 36, 40, 45]. Due to its strong generalizability, we employ multi-task RL for the pre-training process, enabling the pre-trained policy to be quickly fine-tuned for efficient policy development in new tasks.

Finally, our work is related to the multi-agent patrolling problem where the evader often anticipates patrolling strategies and may choose a target or a single path as an action [2, 4, 16, 29]. Conversely, our pursuit-evasion game features simultaneous actions with the evader unaware of the pursuer's real-time locations.

[1]Note that classical heuristic algorithms such as Dijkstra are also less applicable owing to this reason. Moreover, it is less meaningful to assume that there is at least one pursuer at each exit as the pursuer's resources are typically limited [32].

3 PROBLEM FORMULATION

In this section, we first present all the elements for defining the PEGs. Then, we present the state-of-the-art (SOTA) method for solving PEGs. Finally, we give the problem statement of this work.

3.1 Preliminaries

A pursuit-evasion game (PEG) is a two-player game played between a pursuer and an evader, i.e., $N = \{p, e\}$. Following previous works [21, 38, 43], we assume that the pursuer comprises n members denoted as $p = \{1, 2, ..., n\}$, and the pursuer can obtain the real-time location information of the evader with the help of tracking devices. In reality, PEGs are typically played on urban road maps, which can be represented by a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. Let $V' \subset V$ denote the set of exit nodes from which the evader can escape and T the predetermined time horizon of the game. At $t \leq T$, the locations of the evader and pursuer are denoted by l_t^e and $l_t^p = (l_t^1, l_t^2, ..., l_t^n)$, respectively. Then, the history of the game at t is a sequence of past locations of both players, i.e., $h = (l_0^e, l_0^p, ..., l_{t-1}^e, l_{t-1}^p)$. The available action set for both players is the neighboring vertices of the player's current location, i.e., $A_e(h) = \mathcal{N}(l_{t-1}^e)$ and $A_p(h) = \{(l^1, l^2, ..., l^n) | l^i \in \mathcal{N}(l_{t-1}^i), \forall i \in p\}$ where $\mathcal{N}(v)$ denotes the set of neighboring vertices of vertex v. According to the definition of history, we define the information set for each player as the set consisting of indistinguishable histories. As the evader cannot get the pursuer's real-time location information, the information set of the evader is defined as $I_e = \{h | h = (l_0^e, l_0^p, l_1^e, *, ..., l_{t-1}^e, *)\}$, where $*$ represents any possible location of the pursuer. Although the PEG is a simultaneous-move game, we can model it as an extensive-form game (EFG) by assuming that the evader acts first and then the pursuer commits without any information about the evader's current action. The pursuer's information set can be defined as $I_p = \{h | h = (l_0^e, l_0^p, ..., l_{t-1}^e, l_{t-1}^p, *)\}$ since the pursuer knows the evader's location but not the evader's current action.

A behavior policy of a player assigns a probability distribution over the action set for every information set belonging to the player. Notice that the pursuer's action space is combinatorial and expands exponentially with the number of pursuer members. As a result, directly learning a joint policy of the pursuer members would be computationally difficult. To address this issue, instead of learning a joint policy, previous works learn the individual policies either through value decomposition [21] or global critic [20], which are the paradigm of centralized training with decentralized execution (CTDE) for the pursuers. Furthermore, previous works [20] also introduce a new state representation ignoring the game's historical information, which leads to improved performance. In our work, we follow the previously mentioned conventions to define the observations and individual policies for the pursuers.

At each time step t, each pursuer member gets an **observation** $o_t^i = (l_t^p, l_t^e, i, t) \in O^i$, which includes all players' current locations, the id of the pursuer member, and the time step. Each pursuer member i constructs a **policy**[2] $\pi^p : O^i \to \Delta(A_i)$, which assigns a probability distribution over the **action set** $A_i(o_t^i) = \mathcal{N}(l_t^i), \forall i \in p$. As for the evader's policy, we follow previous works [37, 38] that employ High-Level Actions for the evader. That is, the evader only chooses the exit node to escape from and then samples one shortest path from the initial location to the chosen exit node, instead of deciding where to go in the next time step. Specifically, at time step $t = 0$, the evader determines an exit node $v' \in V'$ using the **policy** $\pi^e : V \to \Delta(V')$, samples a shortest path from $l_0^e \in V$ to the chosen exit node v', and then takes actions based on the path.

Here, we give some remarks on the assumption of High-Level Actions of the evader. (i) In our game setting, the evader lacks real-time access to the pursuers' locations, requiring the evader to act without any information about their whereabouts. Therefore, sampling one path for the evader would not lose much information compared with the case where the evader acts step by step. (ii) Training the pursuer against an evader who chooses the shortest path, a worst-case scenario for the pursuer, enhances the robustness of the pursuer's policy. (iii) Though it is a simplification, the problem setting remains highly complex due to the multiple exits and diverse players' initial conditions, enlarging the task space for the pursuer, as detailed in the Introduction and Appendix A Q2.

In summary, given a graph G with the specific set of exit nodes V', the initial locations of the pursuer and evader (l_0^p, l_0^e), and the predetermined time horizon T, we can define a specific PEG as $\mathcal{G} = (G, V', l_0^p, l_0^e, T)$. In the PEG, players will get the non-zero rewards only when the game is terminated. The termination conditions include three cases: (i) the pursuer catches the evader within the time horizon T, i.e., $l_t^e \in l_t^p, t \leq T$; (ii) the evader escapes from an exit node within the time horizon T, i.e., $l_t^e \in V', t \leq T$; (iii) the game reaches the time horizon T. Let $t' \leq T$ be the time step that the game is terminated. Then, for all $t < t'$, $r_t^p = r_t^e = 0$. For $t = t'$, in cases (i) and (iii), the pursuer receives a **reward** $r_t^p = 1$ while the evader incurs a penalty $r_t^e = -1$. In case (ii), the evader gains a reward $r_t^e = 1$, and the pursuer suffers a loss $r_t^p = -1$. Given the exit node chosen by the evader $v' \sim \pi^e$, we have $V^p(\pi^p, v') = \mathbb{E}\big[\sum_{t=0}^{T} r_t^p\big]$ for the pursuer and $V^e(\pi^p, v') = \mathbb{E}\big[\sum_{t=0}^{T} r_t^e\big]$ for the evader, where the expectation is taken over the trajectories induced by π^p. Then, for the policy pair (π^p, π^e), we have $V^p(\pi^p, \pi^e) = \mathbb{E}_{v' \sim \pi^e}\big[V^p(\pi^p, v')\big]$ for the pursuer and $V^e(\pi^p, \pi^e) = \mathbb{E}_{v' \sim \pi^e}\big[V^e(\pi^p, v')\big]$ for the evader.

3.2 Policy-Space Response Oracles

Algorithm 1: PSRO for a specific PEG $\mathcal{G}$

$\Pi_0^p = \{\pi_0^p\}$, $\Pi_0^e = \{\pi_0^e\}$, U_0, σ_0^p, σ_0^e;
for *epoch* $k = 1, 2, \cdots K$ **do**
  Compute the evader's BR policy π_k^e against σ_{k-1}^p;
  Compute the pursuer's BR policy π_k^p against σ_{k-1}^e;
  Expansion: $\Pi_k^p = \Pi_{k-1}^p \cup \{\pi_k^p\}$, $\Pi_k^e = \Pi_{k-1}^e \cup \{\pi_k^e\}$;
  Update meta-game matrix U_k through simulation;
  Compute σ_k^p and σ_k^e using a meta-solver on U_k;
Return: Π_K^p, Π_K^e, σ_K^p, σ_K^e

As one of the popular algorithms, PSRO [18] can be employed to solve a PEG $\mathcal{G}$, shown in Algorithm 1. It commences with each

[2] All the pursuer members share one policy. As the observations include pursuers' ids, different pursuer members can have different behavior [10]. Δ denotes the simplex.

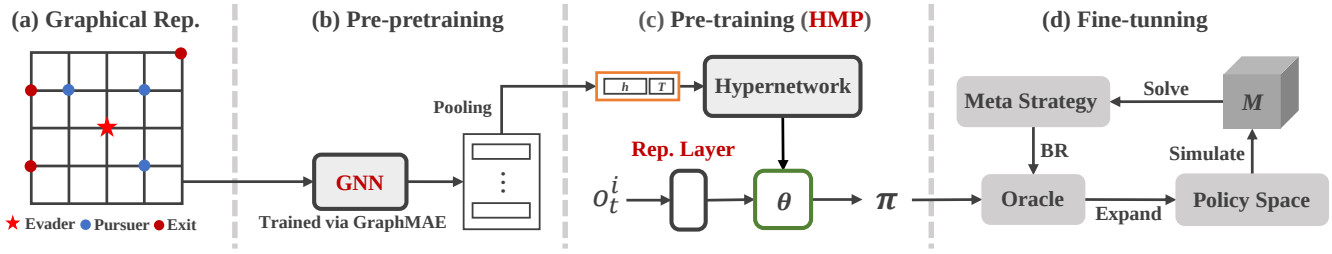


Figure 1: Architecture and training pipeline of Grasper.

player using a random policy (Line 1) and then expands the policy spaces of the pursuer and evader in an iterative manner. At each epoch $1 \leq k \leq K$: (1) Compute the best response (BR) policies of the pursuer π_k^p and evader π_k^e and add them to their policy spaces Π_k^p and Π_k^e (Line 3–5); (2) Construct a meta-game U_k using all policies in each player's policy space (Line 6); (3) Compute the meta-strategy of the pursuer $\sigma_k^p \in \Delta(\Pi_k^p)$ and evader $\sigma_k^e \in \Delta(\Pi_k^e)$ using a meta-solver (e.g., PRD [18]) on the meta-game U_k (Line 7). These processes are repeated for K epochs and then output the final meta-strategy across the players' policy spaces (Line 9).

As the evader's policy is a probability distribution over exit nodes, to compute the BR policy (Line 3), we only need to estimate the value of each exit node through simulations, i.e., $V^e(v') = \mathbb{E}_{\pi^p \sim \sigma_{k-1}^p}\left[V^e(\pi^p, v')\right], \forall v' \in V'$. Then, the evader's BR policy is constructed by applying softmax operation on the values of all the exit nodes. For the pursuer, computing the BR policy is to solve the problem $\pi_k^p = \arg\max_{\pi^p} \mathbb{E}_{\pi^e \sim \sigma_{k-1}^e}\left[V^p(\pi^p, \pi^e)\right]$ (Line 4). As there are multiple pursuer members, we can use MAPPO [39] to learn the BR policy. In the traditional PSRO algorithm, the pursuer's BR policy is learned from scratch, i.e., the BR policy is randomly initialized, which is inefficient. To address this issue, recent works integrate the pre-training and fine-tuning paradigm into PSRO to improve learning efficiency [20]. Specifically, before running PSRO, a base pursuer policy is trained through multi-task RL where each task is generated with a randomly initialized evader's policy. Then, at each PSRO epoch, the pursuer's BR policy is initialized with the pre-trained base policy, rather than learning from scratch, which can largely improve the learning efficiency of the PSRO algorithm.

3.3 Problem Statement

Although PSRO has been successfully applied to solve PEGs, unfortunately, previous works typically focus on solving a specific PEG with predetermined initial conditions which are not always fixed in real-world scenarios: (i) The initial locations of the pursuers and the evader (l_0^p, l_0^e) are not always fixed since attacks (thieves, crimes, terrorists) can occur at any time and location in a city; (ii) The locations of the exit nodes V' may change due to temporary closures and openings; (iii) The time horizon T might vary, as the time required to pursue the evader is not always the same. When any of the initial conditions change, the PEG adapts accordingly. As a consequence, current algorithms can only solve the modified PEG from scratch, leading to significant time consumption and inefficiency. Even the SOTA method presented in the previous section – PSRO with pre-trained base pursuer policy – still suffers from such an issue as the base policy is pre-trained under the premise that the initial condition of the PEG is fixed. In other words, a new base policy must be pre-trained from scratch for the modified PEG since the original base policy may not be a good starting point for the pursuer's BR policy in the modified game (even worse than a randomly initialized BR policy). In this paper, we aim to address this issue by developing a generalist pursuer capable of learning and adapting to different PEGs with varying initial conditions without the need to restart the training process from the beginning.

4 GRASPER

In this section, we introduce Grasper, illustrated in Figure 1. We first present the architecture of Grasper including several innovative components, and then the training pipeline which consists of three stages to efficiently train the networks of Grasper.

4.1 Architecture

4.1.1 ***Graphical Representations of PEGs.*** To generate the pursuer's policy based on the specific PEG, we propose to take the specific PEG as an input of a neural network. To this end, we encode the initial conditions of a PEG except for T into a graph (Figure 1(a)). The time horizon T can be directly fed into the neural network. Specifically, given a PEG $\mathcal{G} = (G, V', l_0^p, l_0^e, T)$, these initial conditions V', l_0^p and l_0^e can be encoded into the graph G by associating each node of the graph with a vector consisting of the following parts: (i) a binary bit $\{0, 1\}$ where 1 indicates that the node is an exit, (ii) a binary bit $\{0, 1\}$ where 1 signifies that the evader's initial location is this node, (iii) the number of pursuers on this node $\{0, \ldots, n\}$ (the total number of pursuers across all nodes equals to n), and (iv) additional information regarding the topology of the graph, such as the degree of the node. This provides a universal representation of any PEG with any initial condition.

4.1.2 ***Game-conditional Base Policies Generation.*** After representing a PEG as a graph, it is natural to leverage a graph neural network (GNN) to encode the PEG with the given initial conditions into a hidden vector. As shown in Figure 1(b), we first feed the graphical representation of the initial conditions into the GNN and get the representations of all the nodes of the graph. Then, we use a pooling operation to integrate all the node representations into a hidden vector which will be concatenated with the time horizon T to get the final representation of the PEG. Next, to generate a base policy conditional on the PEG, we introduce a hypernetwork [12], a neural network that takes the final representation of the PEG as

input and outputs the parameters (weights and biases) of the policy network (Figure 1(c)). Finally, the base policy network serves as a starting point for the training of the pursuer's best response policy in each iteration of the PSRO algorithm (Figure 1(d)).

4.1.3 ***Observation Representation Layer.*** As described earlier, the pursuer's policy is a mapping that associates each observation with a probability distribution over the available action set. Notably, an observation consists of the positions of both players. Representing these positions by mere index numbers of vertices in the graph does not provide much useful information for training, though. Therefore, we seek a more compact and meaningful representation of these observations. Previous works [20, 38] typically train a node embedding model for this purpose. Unfortunately, such a model is often tailored and trained for a specific graph, limiting its generalizability to other graphs. This lack of generalizability makes this method unsuitable for our problem. To address this issue, we adopt a representation layer to encode the pursuer's observations, an approach that is not limited to a specific graph.

As given in Section 3.1, the pursuer's observation $o_t^i = (l_t^p, l_t^e, i, t)$ includes three parts: the players' current locations (l_t^p, l_t^e), the pursuer member's id i, and the current time step t. Thus, the representation layer consists of three components, each of which is a "torch.nn.Embedding" which has been extensively used to encode an integer to a compact representation. The outputs of the three components are concatenated to obtain the representation of the pursuer's observation. This representation layer will be trained jointly with the hypernetwork during the pre-training process. Please refer to Appendix B for details on the architecture of the representation layer, the GNN, and the hypernetwork.

Intuitively, the generalization ability of Grasper benefits from several designs of our architecture. First, the graphical representation offers a universal representation of any PEG, regardless of the graph's topology. Second, GNN can encode different PEGs into fixed-size hidden vectors, which can be directly fed into the hypernetwork (otherwise, additional techniques are required if the sizes of the hidden vectors are varied). Finally, the hypernetwork is designed to generate a specialized policy tailored to a given PEG.

4.2 Training Pipeline

Now we introduce the training pipeline of Grasper, which involves three stages: pre-pretraining, pre-training, and fine-tuning. Prior to delving into the specifics, we first describe how the training set is generated. The training set $\mathcal{I}$ should consist of different PEGs for training. To this end, we generate the training set by randomizing the initial conditions, denoted by (G, V', l_0^p, l_0^e, T). However, this approach may yield certain games that lack meaningful training value. For example, when the evader's initial location is in such close proximity to the exit nodes that the pursuer becomes incapable of capturing the evader regardless of its movements. To exclude these trivial cases, we introduce a filter condition when generating the training set: the shortest path from the evader's initial location to any exit nodes must exceed a predetermined length.

4.2.1 ***Stage I: Pre-pretraining.*** As the hypernetwork takes a feature vector as input, we first use a GNN to encode the graphical representation of the PEG into a fixed-size hidden vector. As the GNN is solely employed to derive the effective representation from the PEG's graphical interpretation, we introduce a pre-pretraining stage (Figure 1(b)) to pre-train the GNN before the actual pre-training stage. This approach is more efficient compared to jointly training the GNN and hypernetwork in the pre-training stage. Specifically, for each game in the training set $\mathcal{G} \in \mathcal{I}$, let $\boldsymbol{A}_{\mathcal{G}}$ and $\boldsymbol{X}_{\mathcal{G}}$ denote the adjacency matrix and feature matrix of the underlying graph, respectively. We first obtain the latent code matrix $\boldsymbol{H}_{\mathcal{G}} = f_{\text{GNN}}(\boldsymbol{X}_{\mathcal{G}}, \boldsymbol{A}_{\mathcal{G}})$ by the GNN and train the GNN via any self-supervised graph learning method (we use the recent SOTA method, GraphMAE [15]). Then, we get the hidden vector by pooling the latent code matrix $\boldsymbol{h}_{\mathcal{G}} = pool(\boldsymbol{H}_{\mathcal{G}})$, which will be fed into the hypernetwork.

Algorithm 2: Pre-training

```
1  Initialize Grasper and the episode buffer D ← ∅;
2  for train epoch = 1, 2, ··· do
3      Uniformly sample c1 games from the training set I;
4      for each of the c1 games G do
5          Randomly generate c2 evader's policies;
6          Generate pursuer's policy π_θ^p ← Grasper(G);
7          for each of the c2 evader's policies π^e do
8              Sample data using π^e, π_θ^p, and π̂^p;
9              Add the data into the episode buffer D;
10     Train the networks by optimizing the loss function L;
11     Clear the episode buffer D ← ∅;
```

4.2.2 ***Stage II: Pre-training.*** Given a fixed evader's policy in a specific PEG, computing the pursuer's best response policy can be seen as an RL task. Thus, we can apply the multi-task RL algorithm to guide the pre-training process, which is shown in Algorithm 2. Different from previous work [20], in these RL tasks, except for the change in the evader's policy, the game's initial conditions also change. To obtain these RL tasks for pre-training, we first randomly sample c_1 games from the training set (Line 3), and then for each game, we randomly sample c_2 evader's policies (Line 5). Once the game and the evader's policy are fixed, the RL task is generated. During pre-training, for each game, we first feed the hidden vector of the game (obtained by the trained GNN) and the time horizon into the hypernetwork to generate the pursuer's base policy, and then for each evader's policy, we collect the training data using the pursuer's base policy (accompany by the representation layer) into the episode buffer. Finally, we train the hypernetwork and the representation layer jointly based on the episode buffer (Lines 6-9). To deal with the multiple pursuer members cases, we employ MAPPO [39] as the underlying RL algorithm.

However, we found that simply applying the MAPPO under the multi-task learning framework can result in low efficiency due to random exploration in the environment. To clarify, consider the example illustrated in Figure 2, which shows the need for a more efficient pre-training method. Assume that the evader's policy is to take the shortest path to one of the exits (denoted by the red path). If the pursuer explores the environment randomly (the orange path), it will probably lose the game and then receive a negative reward. This situation can occur frequently at the beginning of the pre-training process because the pursuer's initial policy is invariably

random. To mitigate this exploration inefficiency[3], we propose a novel scheme: heuristic-guided multi-task pre-training (HMP).

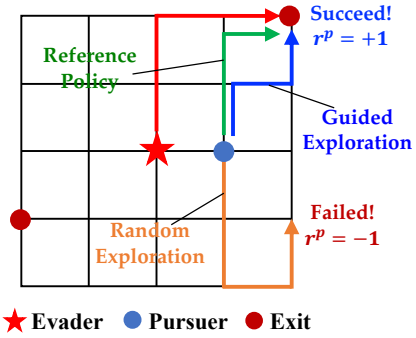


Figure 2: Illustration of HMP.

Note that in the RL tasks used for pre-training, we can acquire the evader's policy, which can be used to guide the exploration of the pursuer's policy. Specifically, given the exit node chosen by the evader's policy π^e, we first induce a reference policy $\hat{\pi}^p$ (represented by the green path) for the pursuer using heuristic methods such as the Dijkstra algorithm. Then, apart from the actions sampled by the generated policy π^p_θ (Line 6), we also sample the reference actions using the reference policy $\hat{\pi}^p$ and add the data to the training buffer (Lines 8-9). Let $L(\boldsymbol{\theta})$ denote the original loss function for training the actor in the MAPPO algorithm. The HMP is implemented by introducing an additional loss into the original loss function: $L = L(\boldsymbol{\theta}) + \alpha \mathrm{KL}(\pi^p \| \hat{\pi}^p)$ where $\alpha \in [0, 1]$ controls the weight of the guidance of the reference policy and KL represents the Kullback–Leibler divergence (for the reference policy $\hat{\pi}^p$, the action probability distribution is obtained by setting the probability of the reference action to 1 while all others to 0).

4.2.3 ***Stage III: Fine-tuning.*** In this phase, we integrate the pre-trained pursuer policy into the PSRO framework, as shown in Algorithm 3. The pursuer's policy π^p_0 is initialized using the output neural network from the pre-trained Grasper, which takes the graphical representation of the specific PEG as an input. Simultaneously, the evader's policy π^e_0 is randomly initialized (Line 2). Then we follow the standard PSRO framework: in each iteration k, the best response (BR) policies for both players, π^p_k and π^e_k, are computed using their respective BR oracles (Lines 4-6). These BR policies are then added to the policy sets Π^p_k and Π^e_k (Line 7), and the meta-game matrix U_k is updated through simulation (Line 8). Finally, the meta distribution (σ^p_k, σ^e_k) is computed using any meta-solver (Line 9).

The BR oracles for both players are the important components of the PSRO algorithm. The evader's BR oracle follows the standard PSRO algorithm given in Algorithm 1. The key difference between our fine-tuning process and the standard PSRO algorithm lies in the training of the pursuer's BR policy, which is highlighted in blue. Specifically, given the pre-trained policy π^p_0 conditional to the initial conditions, we can use it as the starting point for the computation of the pursuer's BR policy (Line 5), rather than training from scratch. This allows us to simply fine-tune the pre-trained policy π^p_0 over a few episodes (Line 6) to quickly obtain the BR policy, significantly enhancing the learning efficiency.

Algorithm 3: Fine-tuning

1 **Require**: Grasper, PEG $\mathcal{G}$;
2 $\Pi^p_0 = \{\pi^p_0 \leftarrow \text{Grasper}(\mathcal{G})\}$, $\Pi^e_0 = \{\pi^e_0\}$, U_0, σ^p_0, σ^e_0;
3 **for** *epoch* $k = 1, 2, \cdots K$ **do**
4 Compute the evader's BR policy π^e_k against σ^p_{k-1};
5 Initialize the pursuer's BR policy $\pi^p_k \leftarrow \pi^p_0$;
6 Train π^p_k against σ^e_{k-1} for few episodes;
7 Expansion: $\Pi^p_k = \Pi^p_{k-1} \cup \{\pi^p_k\}$, $\Pi^e_k = \Pi^e_{k-1} \cup \{\pi^e_k\}$;
8 Update meta-game matrix U_k through simulation;
9 Compute σ^p_k and σ^e_k using a meta-solver on U_k;
10 **Return**: Π^p_K, Π^e_K, σ^p_K, σ^e_K

5 EXPERIMENTS

In this section, we perform experiments to evaluate the performance of Grasper and the effectiveness of different components[4].

5.1 Setup

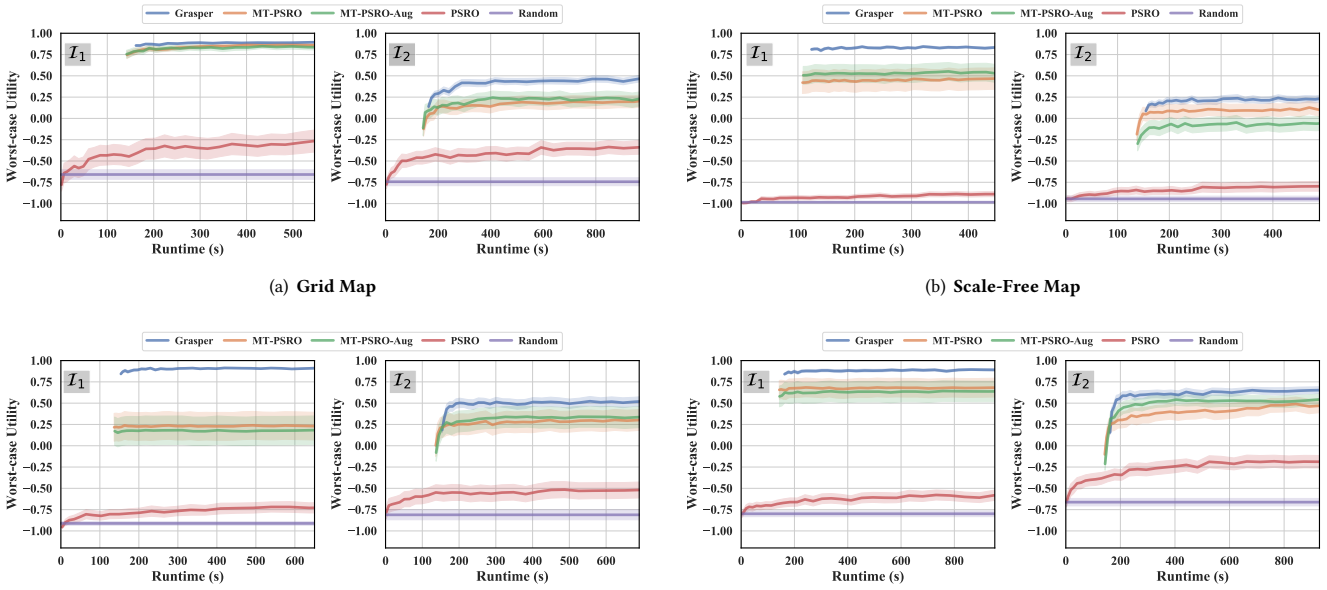
Hyperparameters. The number of pursuers is $n = 5$, the number of exit nodes is 8, the time horizon T is $6 \le T \le 10$, and the number of pre-training episodes is 20 million (20M). For PSRO, the number of episodes used for training the best response is 10. We conduct experiments on four maps: the grid map with size 10×10, the scale-free graph with 300 nodes, the Singapore map [38] with 372 nodes, and the Scotland-Yard map [28] with 200 nodes. To simulate the situation where a road might be temporally blocked due to congestion or traffic accidents, we set the probability of an edge between two nodes to 0.8 for the grid map, 0.9 for the Singapore map, and 1.0 (i.e., no congestion) for the other two maps. More details on the hyperparameters can be found in Appendix B.

Worst-case Utility. Given that a PEG is a zero-sum game, we use the pursuer's worst-case utility (as the evader always chooses the shortest path from the initial location to the chosen exit) to measure the quality of the solution: $u^p = \mathbb{E}_{\pi^p \sim \sigma^p, \pi^e \sim \sigma^e} \mathbb{E}[r^p]$, where the inner expectation is taken over the trajectories induced by π^p and π^e which are respectively sampled according to σ^p and σ^e.

Training and Test Sets. (1) We generate $|\mathcal{I}| = 1000$ games as the training set. During the generation, the minimum length of the evader's shortest path is set to 6 for the grid map and 5 for other maps. (2) We create two test sets, $\mathcal{I}_1$ and $\mathcal{I}_2$, each containing 30 games. (i) $\mathcal{I}_1$ includes the games sampled from the training set $\mathcal{I}_1 \subset \mathcal{I}$ (in-distribution test). (ii) $\mathcal{I}_2$ contains the games distinct from the training set $\mathcal{I}_2 \cap \mathcal{I} = \emptyset$ (out-of-distribution test). To avoid trivial cases (the games that are either too difficult or too simple for the pursuers), we constrain the zero-shot performance of Grasper (i.e., the worst-case utility of the generated policy without fine-tuning) within the range: $[0.8, 0.9]$ for $\mathcal{I}_1$ and $[0.1, 0.2]$ for $\mathcal{I}_2$.

[3]Notice that many exploration methods in RL such as RND [5] typically encourage the policy to explore novel states of the environment, which are different from our design where random exploration is less favored.

[4]The code is available at https://github.com/IpadLi/Grasper.



(a) **Grid Map**

(b) **Scale-Free Map**

(c) **Singapore Map**

(d) **Scotland-Yard Map**

Figure 3: Evaluation performance. The shaded area represents the standard error.

Baselines. (i) Multi-task PSRO (MT-PSRO): the state-of-the-art (SOTA) approach adapted from [20], which also uses the observation representation layer and HMP. (ii) MT-PSRO with augmentation (MT-PSRO-Aug): the hidden vector obtained from the pre-trained GNN and the time horizon are concatenated to the output of the observation representation layer. (iii) PSRO: the standard PSRO method. (iv) Random: the pursuer randomly selects actions. The relation between different methods is given in Appendix B.

5.2 Results

The experimental results are summarized in Figure 3. The x-axis is the running time. For the purpose of a fair comparison, apart from the running time of the fine-tuning stage (the PSRO procedure), we also include the running time of pre-pretraining and pre-training (called the pre-training time for convenience). Since the games in the training set are uniformly randomly sampled during pre-training, we approximate the pre-training time of each game by averaging the total pre-training time over the training set. Then, for each testing game, we add the pre-training time to the running time[5] (the horizontal gap between 0 and the start of the line). From the results, we can draw several conclusions as follows.

(i) Given a fixed number of episodes for the fine-tuning process, Grasper can start from and converge to a higher average worst-case utility than the baselines, although it takes a certain pre-training time, demonstrating the effectiveness of the pre-pretraining and pre-training in accelerating the PSRO procedure. Note that MT-PSRO and MT-PSRO-Aug also employ pre-pretraining or pre-training, but they perform worse than Grasper, showcasing the superiority of Grasper. (ii) For a fair comparison, MT-PSRO-Aug also integrates the information about the initial conditions of the PEGs. The results clearly show the necessity of the hypernetwork in Grasper. This can be also partly verified by comparing MT-PSRO and MT-PSRO-Aug where their performance is comparable, meaning that naively integrating the information about the initial conditions does not bring much benefit and novel designs are necessary. (iii) An interesting result is that even on the test set $\mathcal{I}_1$ (in-distribution test), MT-PSRO and MT-PSRO-Aug, the strongest baselines, perform worse on all the other maps than on the grid map. We hypothesize the reason is that the other maps are more heterogeneous than the grid map. For example, the degree of the nodes varies from 1 to 16 in the Singapore map while it remains between 2 to 4 in the grid map. Thus, the games generated on the Singapore map share much less similarity. In this sense, the information about the initial conditions is particularly important when solving different PEGs. (iv) In all cases, the performance of Grasper is much more stable than the baselines (smaller standard error) as Grasper can generate distinct policies for different PEGs. In contrast, other baselines either entirely ignore or naively integrate the information about the initial conditions of the PEGs, which renders them hard to generalize to different PEGs, leading to larger performance variance than Grasper. (v) The results on the test set $\mathcal{I}_2$ show that Grasper can solve unseen games, exhibiting better generalizability than the baselines.

5.3 Ablations

In this section and Appendix C, we perform ablation studies to show the effectiveness of different components, which further deepens the understanding of our proposed framework.

Effectiveness of Different Modules. First, we study the contribution of HMP and the observation representation layer (Rep.) to

[5]Note that the amortized pre-training time for Grasper, MT-PSRO, and MT-PSRO-Aug is similar as the *pre-pretraining time* is very short as shown in Table 2.

the performance of Grasper, as shown in Table 1. The results show that we can get better performance (high worst-case utility and small standard error) only when combining the two components, meaning that both two components are indispensable for Grasper.

Table 1: Ablation studies. The results are obtained in the grid map. ✓ means the module is used.

	HMP	Rep.	Utility
$\mathcal{I}_1$	✓	✓	**0.90 ± 0.01**
	✓		−0.54 ± 0.06
		✓	−0.05 ± 0.17
			−0.52 ± 0.08
$\mathcal{I}_2$	✓	✓	**0.45 ± 0.04**
	✓		−0.60 ± 0.06
		✓	−0.64 ± 0.11
			−0.63 ± 0.06

Effectiveness of Pre-pretraining. Next, we investigate the effectiveness of the pre-pretraining stage in accelerating the whole training procedure of Grasper. Since jointly training GNN and the other parts of Grasper for 20M pre-training episodes requires a long running time, in this ablation study, we focus on the first 2M pre-training episodes and compare the running time of Grasper with pre-pretraining (w/ PP) and without pre-pretraining (w/o PP). The training curves are shown in Figure 4, which shows that using pre-pretraining can significantly accelerate the training procedure (3.9 times faster than without using pre-pretraining).

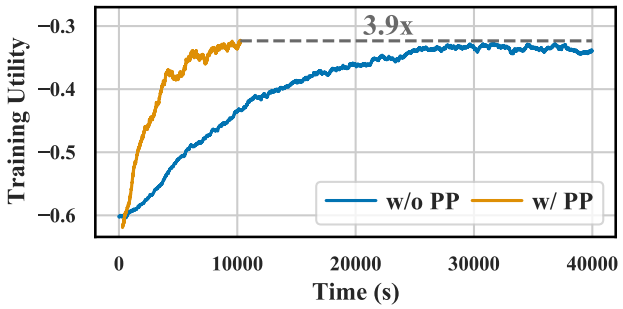


Figure 4: Pre-training curves.

The quantitative values of the running time of the pre-pretraining and pre-training are given in Table 2. As the pre-pretraining time (304.2 seconds) is much shorter than the pre-training time (9954.9 seconds), the curves of Grasper, MT-PSRO, and MT-PSRO-Aug shown in Figure 3 start from a similar position in the x-axis.

Influence of Evader's Initial Location. We perform some experiments using Grasper to provide some insights into the PEG. In Figure 5, we present the pursuer's utility when the evader randomizes the initial location over the grid map. We found that in some areas the pursuers can have high utility. For example, in the top-right of the left figure, there are three pursuers and only one exit, which means it could be hard for the evader to escape. In the bottom-right of the right figure, as the pursuer's initial location is near the two exits, it could be easy for the pursuer to catch the evader, even though there is only one pursuer in this area. The results reflect the intuition that Grasper can generate distinct policies for different games and hence, the performance is more stable.

Table 2: Running time (second).

	w/o PP	w/ PP
Pre-pretraining	N/A	304.2
Pre-training	39977.3	9954.9
Total	39977.3	10259.1 (**3.9x**)

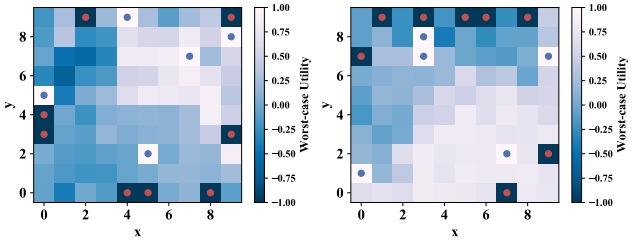


Figure 5: Zero-shot worst-case utility of the pursuer for each possible evader's initial location. Red dots are exits and blue dots are pursuers' initial locations.

6 CONCLUSIONS

In this work, we investigate how to efficiently solve different PEGs with varying initial conditions. First, we propose a novel generalizable framework, Grasper, which includes several critical components: (i) a GNN to encode a specific PEG into a hidden vector, (ii) a hypernetwork to generate the base policies for the pursuers conditional on the hidden vector and time horizon, (iii) an observation representation layer to encode the pursuers' observations into compact and meaningful representations. Second, we introduce an efficient three-stage training method which includes: (i) a pre-pretraining stage that learns robust PEG representations through GraphMAE, (ii) a heuristic-guided multi-task pre-training stage that leverages a reference policy derived from heuristic methods such as Dijkstra to regularize pursuer policies, and (iii) a fine-tuning stage that utilizes PSRO to generate pursuer policies on designated PEGs. Finally, extensive experiments demonstrate the superiority of Grasper over baselines in terms of solution quality and generalizability. To the best of our knowledge, this is the first attempt to consider the generalization problem in the domain of PEGs. There are some limitations that we will investigate in future works. (i) In the current pre-training approach, tasks are uniformly randomly sampled, which could be less efficient. We will consider more efficient task sampling strategies for pre-training, e.g., AdA [1]. (ii) We will consider a model capable of generalizing to different PEGs with different underlying graph topologies, e.g., generalizing from grid maps to scale-free maps. (iii) We will consider a model capable of tackling more complex settings, e.g., learning-based evader.

ACKNOWLEDGMENTS

This research is supported by the National Research Foundation, Singapore under its Industry Alignment Fund – Pre-positioning (IAF-PP) Funding Initiative. Any opinions, findings and conclusions, or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore. Youzhi Zhang is supported by the InnoHK Fund. 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. The content is solely the responsibility of the authors and does not necessarily represent the official views of the funding agencies.

REFERENCES

[1] Adaptive Agent Team, Jakob Bauer, Kate Baumli, Satinder Baveja, Feryal Behbahani, Avishkar Bhoopchand, Nathalie Bradley-Schmieg, Michael Chang, Natalie Clay, Adrian Collister, Vibhavari Dasagi, Lucy Gonzalez, Karol Gregor, Edward Hughes, Sheleem Kashem, Maria Loks-Thompson, Hannah Openshaw, Jack Parker-Holder, Shreya Pathak, Nicolas Perez-Nieves, Nemanja Rakicevic, Tim Rocktäschel, Yannick Schroecker, Jakub Sygnowski, Karl Tuyls, Sarah York, Alexander Zacherl, and Lei Zhang. 2023. Human-timescale adaptation in an open-ended task space. *arXiv preprint arXiv:2301.07608* (2023).

[2] Noa Agmon, Gal A Kaminka, and Sarit Kraus. 2011. Multi-robot adversarial patrolling: facing a full-knowledge opponent. *Journal of Artificial Intelligence Research* 42 (2011), 887–916.

[3] Shaunak D Bopardikar, Francesco Bullo, and Joao P Hespanha. 2008. On discrete-time pursuit-evasion games with sensing limitations. *IEEE Transactions on Robotics* 24, 6 (2008), 1429–1439.

[4] Jan Buermann and Jie Zhang. 2022. Multi-robot adversarial patrolling strategies via lattice paths. *Artificial Intelligence* 311 (2022), 103769.

[5] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. 2018. Exploration by random network distillation. In *ICLR*.

[6] Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*. 160–167.

[7] Zhijian Duan, Wenhan Huang, Dinghuai Zhang, Yali Du, Jun Wang, Yaodong Yang, and Xiaotie Deng. 2023. Is Nash equilibrium approximator learnable?. In *AAMAS*. 233–241.

[8] Zhijian Duan, Yunxuan Ma, and Xiaotie Deng. 2023. Are equivariant equilibrium approximators beneficial? *arXiv preprint arXiv:2301.11481* (2023).

[9] Xidong Feng, Oliver Slumbers, Ziyu Wan, Bo Liu, Stephen McAleer, Ying Wen, Jun Wang, and Yaodong Yang. 2021. Neural auto-curricula in two-player zero-sum games. In *NeurIPS*. 3504–3517.

[10] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2018. Counterfactual multi-agent policy gradients. In *AAAI*. 2974–2982.

[11] Ross Girshick. 2015. Fast R-CNN. In *ICCV*. 1440–1448.

[12] David Ha, Andrew M. Dai, and Quoc V. Le. 2017. HyperNetworks. In *ICLR*.

[13] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. 2022. Masked autoencoders are scalable vision learners. In *CVPR*. 16000–16009.

[14] Karel Horák and Branislav Bošanskỳ. 2017. Dynamic programming for one-sided partially observable pursuit-evasion games. In *ICAART*. 503–510.

[15] Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. 2022. GraphMAE: self-supervised masked graph autoencoders. In *KDD*. 594–604.

[16] Li Huang, MengChu Zhou, Kuangrong Hao, and Edwin Hou. 2019. A survey of multi-robot regular and adversarial patrolling. *IEEE/CAA Journal of Automatica Sinica* 6, 4 (2019), 894–903.

[17] Linan Huang and Quanyan Zhu. 2021. A dynamic game framework for rational and persistent robot deception with an application to deceptive pursuit-evasion. *IEEE Transactions on Automation Science and Engineering* 19, 4 (2021), 2918–2932.

[18] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. 2017. A unified game-theoretic approach to multiagent reinforcement learning. In *NeurIPS*. 4190–4203.

[19] Pengdeng Li, Xinrun Wang, Shuxin Li, Hau Chan, and Bo An. 2023. Population-size-aware policy optimization for mean-field games. In *ICLR*.

[20] Shuxin Li, Xinrun Wang, Youzhi Zhang, Wanqi Xue, Jakub Černý, and Bo An. 2023. Solving large-scale pursuit-evasion games using pre-trained strategies. In *AAAI*. 11586–11594.

[21] Shuxin Li, Youzhi Zhang, Xinrun Wang, Wanqi Xue, and Bo An. 2021. CFR-MIX: Solving imperfect information extensive-form games with combinatorial action space. In *IJCAI*. 3663–3669.

[22] Xiuxian Li, Min Meng, Yiguang Hong, and Jie Chen. 2022. A survey of decision making in adversarial games. *arXiv preprint arXiv:2207.07971* (2022).

[23] Victor G Lopez, Frank L Lewis, Yan Wan, Edgar N Sanchez, and Lingling Fan. 2019. Solutions for multiagent pursuit-evasion games on communication graphs: Finite-time capture and asymptotic behaviors. *IEEE Transactions on Automatic Control* 65, 5 (2019), 1911–1923.

[24] Luke Marris, Ian Gemp, Thomas Anthony, Andrea Tacchetti, Siqi Liu, and Karl Tuyls. 2022. Turbocharging solution concepts: Solving NEs, CEs and CCEs with neural equilibrium solvers. In *NeurIPS*. 5586–5600.

[25] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. 2020. GCC: Graph contrastive coding for graph neural network pre-training. In *SIGKDD*. 1150–1160.

[26] Frederick P Rivara and Christopher D Mack. 2004. Motor vehicle crash deaths related to police pursuits in the United States. *Injury Prevention* 10, 2 (2004), 93–95.

[27] Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098* (2017).

[28] Martin Schmid, Matej Moravčík, Neil Burch, Rudolf Kadlec, Josh Davidson, Kevin Waugh, Nolan Bard, Finbarr Timbers, Marc Lanctot, G Zacharias Holland, Davoodi Davoodi, Alden Christianson, and Michael Bowling. 2023. Student of Games: a unified learning algorithm for both perfect and imperfect information games. *Science Advances* 9, 46 (2023), eadg3256.

[29] Efrat Sless, Noa Agmon, and Sarit Kraus. 2014. Multi-robot adversarial patrolling: facing coordinated attacks. In *AAMAS*. 1093–1100.

[30] Milind Tambe. 2011. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press.

[31] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Mehdi Azabou, Eva L Dyer, Remi Munos, Petar Veličković, and Michal Valko. 2022. Large-scale representation learning on graphs via bootstrapping. In *ICLR*.

[32] Jason Tsai, Zhengyu Yin, Jun-young Kwak, David Kempe, Christopher Kiekintveld, and Milind Tambe. 2010. Urban security: Game-theoretic resource allocation in networked domains. In *AAAI*. 881–886.

[33] Rene Vidal, Omid Shakernia, H Jin Kim, David Hyunchul Shim, and Shankar Sastry. 2002. Probabilistic pursuit-evasion games: Theory, implementation, and experimental evaluation. *IEEE Transactions on Robotics and Automation* 18, 5 (2002), 662–669.

[34] Tung-Long Vuong, Do-Van Nguyen, Tai-Long Nguyen, Cong-Minh Bui, Hai-Dang Kieu, Viet-Cuong Ta, Quoc-Long Tran, and Thanh-Ha Le. 2019. Sharing experience in multitask reinforcement learning. In *IJCAI*. 3642–3648.

[35] Yuanda Wang, Lu Dong, and Changyin Sun. 2020. Cooperative control for multi-player pursuit-evasion games with reinforcement learning. *Neurocomputing* 412 (2020), 101–114.

[36] Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. 2007. Multi-task reinforcement learning: A hierarchical Bayesian approach. In *ICML*. 1015–1022.

[37] Wanqi Xue, Bo An, and Chai Kiat Yeo. 2022. NSGZero: efficiently learning non-exploitable policy in large-scale network security games with neural Monte Carlo tree search. In *AAAI*. 4646–4653.

[38] Wanqi Xue, Youzhi Zhang, Shuxin Li, Xinrun Wang, Bo An, and Chai Kiat Yeo. 2021. Solving large-scale extensive-form network security games via neural fictitious self-play. In *IJCAI*. 3713–3720.

[39] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. 2022. The surprising effectiveness of PPO in cooperative multi-agent games. In *NeurIPS Datasets and Benchmarks Track*. 24611–24624.

[40] Sihan Zeng, Malik Aqeel Anwar, Thinh T Doan, Arijit Raychowdhury, and Justin Romberg. 2021. A decentralized policy gradient approach to multi-task reinforcement learning. In *UAI*. 1002–1012.

[41] Hengrui Zhang, Qitian Wu, Junchi Yan, David Wipf, and Philip S Yu. 2021. From canonical correlation analysis to self-supervised graph neural networks. In *NeurIPS*. 76–89.

[42] Youzhi Zhang, Bo An, Long Tran-Thanh, Zhen Wang, Jiarui Gan, and Nicholas R Jennings. 2017. Optimal escape interdiction on transportation networks. In *IJCAI*. 3936–3944.

[43] Youzhi Zhang, Qingyu Guo, Bo An, Long Tran-Thanh, and Nicholas R Jennings. 2019. Optimal interdiction of urban criminals with the aid of real-time information. In *AAAI*. 1262–1269.

[44] Yu Zhang and Qiang Yang. 2021. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering* 34, 12 (2021), 5586–5609.

[45] Mandi Zhao, Pieter Abbeel, and Stephen James. 2022. On the effectiveness of fine-tuning versus meta-reinforcement learning. In *NeurIPS*. 26519–26531.

[46] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2021. Graph contrastive learning with adaptive augmentation. In *WWW*. 2069–2080.

[47] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. 2008. Regret minimization in games with incomplete information. In *NeurIPS*. 1729–1736.