2024 American Control Conference (ACC)
July 8-12, 2024. Toronto, Canada

# Solving Two-Player General-Sum Game Between Swarms

Mukesh Ghimire[1], Lei Zhang[1], Wenlong Zhang[2], Yi Ren[1], and Zhe Xu[1†]

*Abstract*— **Hamilton-Jacobi-Isaacs (HJI) PDEs are the governing equations for the two-player general-sum games. Unlike Reinforcement Learning (RL) methods, which are data-intensive methods for learning value function, learning HJ PDEs provide a guaranteed convergence to the Nash Equilibrium value of the game when it exists. However, a caveat is that solving HJ PDEs becomes intractable when the state dimension increases. To circumvent the curse of dimensionality (CoD), physics-informed machine learning methods with supervision can be used and have been shown to be effective in generating equilibrial policies in two-player general-sum games. In this work, we extend the existing work on agent-level two-player games to a two-player swarm-level game, where two sub-swarms play a general-sum game. We consider the *Kolmogorov forward equation* as the dynamic model for the evolution of the densities of the swarms. Results show that policies generated from the physics-informed neural network (PINN) result in a higher payoff than a Nash Deep Q-Network (Nash DQN) agent and have comparable performance with numerical solvers.**

## I. INTRODUCTION

Swarms, or groups of robots have the ability to carry out complex tasks that are difficult for a single agent. Swarms have been deployed to perform diverse tasks, for example, surveillance and reconnaissance in a military capacity, search and rescue, and even entertainment in the form of light shows [1]. Other applications include task allocations [2], complex formation [3], and decision making [4]. As new applications emerge every day, it is imminent that there will be multiple groups of swarms that may be operating in the same region with their own objective. One of the most trivial examples would be a zero-sum game with two swarm groups with competing objectives. Most of the work in swarms focuses on studying swarm behaviors in a more single-agent setting, where interactions are limited within the swarm group. While some work have proposed game theoretic frameworks to solve problems in communication networks [5], [6], they do so in a distributed fashion. We are interested in finding a high level control strategy for swarms with arbitrarily large population.

Modeling of swarms can be broadly divided into *macroscopic* and *microscopic* models. *Macroscopic* models are invariant to the number of agents whereas individual-level *microscopic* models change with the number of agents [7], and become intractable as the number of agents gets large [8]. As such *macroscopic* models can be thought of as robust models that can be used to study large-scale swarms. A building block for studying such swarms is the *Kolmogorov forward equation* which describes the evolution of the density of a stochastic process. We define density as the ratio of the population of swarms in a region to its total population. Under certain conditions, the *Kolmogorov forward equation* can be applied to encode the macroscopic behavior of a swarm.

Swarms exemplify multi-agent systems [9], with interactions that can be modeled as either zero-sum or general-sum differential games, based on the specific objectives [10]. The Nash Equilibrial values of these games are viscosity solutions to the Hamilton-Jacobi-Isaacs (HJI) equations [11]. However, conventional methods for solving HJI PDEs typically involve mesh-based approaches and encounter challenges like the Curse of Dimensionality (CoD) as the state dimension expands [12]. The CoD is effectively bypassed by employing a neural network, known for its exceptional capability as a universal function approximator [13].

In this work, we introduce a general-sum game framework for swarm groups, characterized by a continuous evolution of density over time. Our contribution lies in extending the existing research on learning and controlling swarm systems from agent-level zero/general-sum games to swarm-level general-sum games. To evaluate the efficacy of our approach, we compare its performance with that of a commonly used reinforcement learning method for games and a numerical solver.

## II. RELATED WORK

**Swarm Control.** Control of swarms is of great interest to the research community. An existing challenge in this line of research is developing models and control mechanisms for large-scale swarms [7]. [8] devised an optimal control strategy for controlling a large-scale swarm to a target distribution. [14] used a leader-follower framework for herding a robotic swarm to a desired distribution. Another work that is in the spirit of our work is [15], which applied Pontryagin's Maximum Principle for control of a large-scale robotic population in an optimal control setting. In addition, [5] and [6] proposed a game-theoretic approach on a graph to solve the *Coverage game*. Another common line of work includes using a Markov chain that models the evolution of the density distribution. [16] used the Markov chain and provided a probabilistic control algorithm for swarms of agents subject to some temporal logic specifications.

This research is partially supported by the National Science Foundation under grants CNS 2304863 and CNS 2339774, and the Office of Naval Research under grant ONR N00014-23-1-2505.

[1]M. Ghimire, L. Zhang, Y. Ren, and Z. Xu are with the School of Engineering, Matter, Transport, and Energy, Arizona State University, Tempe, AZ 85287, USA. Email: {mghimire, lzhan300, yiren, xzhe1}@asu.edu

[2]W. Zhang is with the School of Manufacturing Systems and Networks, Arizona State University, Mesa, AZ 85212, USA. Email: wenlong.zhang@asu.edu

[†]Address all correspondence to this author.

**Solving HJI PDEs using Deep Learning.** Recent works have considered using autoregressive methods such as physics-informed machine learning to learn values of zero-sum [17], and general-sum [10] differential games. [10] extended [17] from zero-sum game with continuous value function to a general-sum game with discontinuous values with respect to states and time. Nash Equilibrium values of general-sum differential games satisfy HJI PDEs, which makes the residual of HJI PDEs a good candidate for a loss function used in physics-informed neural networks. By minimizing the PDE residuals alongside the boundary conditions, a neural network, often a deep one, is trained to predict the Nash equilibrial values associated with the game.

**Multi-Agent Reinforcement Learning (MARL).** MARL, unlike single-agent reinforcement learning, addresses the problem of decision-making involving multiple agents that operate in a common environment. Standard Rollout algorithm is used where the problem is reformulated as a single-agent by using a joint action space [18]. Standard Rollout algorithm has been extended to Multi-agent Rollout algorithm in order to reduce the complexity arising from the joint action space [19]. While most algorithms are in the spirit of optimal control, some works exist that have formulated multi-agent decision-making problems as a game [20], [21]. [20] proposed a modified Q-learning method where Q-values are defined on joint action space and are updated following Nash Equilibrial strategies. [21] extended this idea from a tabular method to a scalable one using the power of deep neural networks. In contrast to the case studies that are discussed in these two related works, our case studies have significantly larger action sets.

This paper is organized as follows. In Section III, we present the notations that appear in the paper frequently. In Section IV, we discuss the fundamental concepts, assumptions, and challenges that are crucial to the development of the paper. We also motivate the problem formulation in this section. In Section V, we discuss the algorithms that are central to the contributions highlighted in the paper. In Section VI, we test our algorithm on different case studies. Finally in Section VII, we conclude with some limitations of the current work and the possible future directions.

## III. NOTATIONS

Borrowing notations from graph theory, we denote a directed graph by the tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ containing a set of $M$ vertices, $\mathcal{V} = \{1, \dots, M\}$, and a set of $N_{\mathcal{E}}$ edges, $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$, where $e = (i, j) \in \mathcal{E}$ if there is an edge from vertex $i \in \mathcal{V}$ to vertex $j \in \mathcal{V}$. A source map is defined as $S : \mathcal{E} \to \mathcal{V}$ and a target map as $T : \mathcal{E} \to \mathcal{V}$ for which $S(e) = i$ and $T(e) = j$, whenever $e = (i, j) \in \mathcal{E}$. The graph $\mathcal{G}$ is said to be *bidirected* if $(v, w) \in \mathcal{E}$ implies that $(w, v) \in \mathcal{E}$ for all $v, w \in \mathcal{V}$.

We follow [10]'s notation system and the implementation therein for HJI equations. Let $\mathcal{X}_i$ and $\mathcal{U}_i$ denote the state and action space respectively for Player $i$. The time-invariant state dynamics of Player $i$ is denoted by $\dot{\boldsymbol{x}}_i(t) = \boldsymbol{f}(\boldsymbol{x}_i(t), \boldsymbol{u}_i(t))$ for $\boldsymbol{x}_i(t) \in \mathcal{X}_i$ and $\boldsymbol{u}_i \in \mathcal{U}_i$. Given a time horizon $T$, the instantaneous and terminal losses of Player $i$ are denoted by $l$, and $c(\boldsymbol{x}_i, \boldsymbol{x}_{-i})$ respectively. Note that $l$ is a constant in this formulation, however, in general, it is a function of state $(l(\boldsymbol{x}_i, \boldsymbol{x}_{-i}))$. For a complete-information general-sum differential game between two players, the value function for each player is $\nu_i(\cdot, \cdot, \cdot) : \mathcal{X}_i \times \mathcal{X}_{-i} \times [0, T] \to \mathbb{R}$. We will adopt shorthands $\boldsymbol{f}_i$, $l_i$, $c_i$, and $\nu_i$ respectively to denote player-wise dynamics, losses, and the value. We use $\mathbf{a}_i = (a_i, a_{-i})$ to concatenate ego and the other player's variables. $\nabla_x$ denotes partial derivative with respect to $x$.

## IV. PRELIMINARIES

**Swarm Modeling.** We consider two homogeneous sub-swarms, each containing $N$ agents that occupy the regions or vertices $\mathcal{V}$. The swarms evolve in continuous time over this region, which can be denoted by vertices on the graph. We denote the graph by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with vertices $\mathcal{V}$ denoting the regions where swarms reside, and $\mathcal{E}$ representing the edges along which the swarms can transition.

Let, $X_v^i(t)$ be the number of agents of sub-swarm $i \in \{1, 2\}$ in the region $v \in \mathcal{V}$. The fraction (or empirical distribution) of the sub-swarms $i$ at location $v \in \mathcal{V}$ at time $t$ is calculated as $\frac{1}{N} X_v^i(t)$. Let $\Omega = \{\mathbf{y} \in \mathbb{R}^n : \sum_{i=1}^n y_i = 1\}$. As $N \to \infty$, the empirical distribution converges to a deterministic quantity $\boldsymbol{x}(t) \in \Omega$, which can be used as a state of the sub-swarm $i$. We denote the state of the sub-swarm $i$ as a vector $\boldsymbol{x}_i(t)$, whose each entry $x_{i,v}(t)$ denotes the fraction of agents in the region $v \in \mathcal{V}$ at time $t$. We use the *Kolmogorov forward equation*, also known as the *mean-field model* as in [14] to evolve the state of the sub-swarm:

$$\dot{\boldsymbol{x}}(t) = \sum_{e \in \mathcal{E}} u_e(t) \mathbf{B}_e \boldsymbol{x}(t), \quad \boldsymbol{x}(0) = \boldsymbol{x}^0 \in \Omega \tag{1}$$

where $\mathbf{B}_e$ are the control matrices with the following entries

$$B_e^{ij} = \begin{cases} -1 & \text{if } i = j = S(e) \\ 1 & \text{if } i = T(e), j = S(e) \\ 0 & \text{otherwise} \end{cases}$$

**Hamilton-Jacobi-Isaacs Equations.** The Nash equilibrial values of a two-player general-sum differential game, when they exist, solve the following HJI equations $(H)$ and satisfy the following boundary condition $(B)$ [22]:

$$\begin{aligned} H(\nu_i, \nabla_{\mathbf{x}_i} \nu_i, \mathbf{x}_i, t) &:= \nabla_t \nu_i + \nabla_{\mathbf{x}_i}^\top \mathbf{f}_i - l_i = 0 \\ B(\nu_i, \mathbf{x}_i) &:= \nu_i(\mathbf{x}_i, T) - c_i = 0, \quad \text{for } i = 1, 2. \end{aligned} \tag{2}$$

The player's policies for ego agent (and other) are derived by maximizing the equilibrial Hamiltonian $h_i(\mathbf{x}_i, \nabla_{\mathbf{x}_i} \nu_i, t) = \nabla_{\mathbf{x}_i} \nu_i^\top \boldsymbol{f}_i - l_i$; $\boldsymbol{u}_i = \arg\max_{\boldsymbol{u} \in \mathcal{U}_i} \{h_i\}$, and $\boldsymbol{u}_{-i} = \arg\max_{\boldsymbol{u} \in \mathcal{U}_{-i}} \{h_{-i}\}$.

**Pontryagin's Maximum Principle:** We can use the PMP equation to generate open-loop policies, which is often more tractable than solving HJI equations. These open-loop policies can be used as a basis for evaluating the learned closed-loop policies. For a given initial state $(\boldsymbol{x}_1^0, \boldsymbol{x}_2^0) \in \mathcal{X}_1 \times \mathcal{X}_2$, we obtain the open-loop policies by solving the following boundary value problem (BVP) according to PMP:

$$\begin{aligned}
\dot{\boldsymbol{x}}_i &= \boldsymbol{f}_i, \quad \boldsymbol{x}_i[0] = \boldsymbol{x}_i^0, \\
\dot{\boldsymbol{\lambda}}_i &= -\nabla_{\boldsymbol{x}_i} h_i, \quad \boldsymbol{\lambda}_i[T] = -\nabla_{\boldsymbol{x}_i} c_i, \\
\boldsymbol{u}_i &= \arg\max_{\boldsymbol{u} \in \mathcal{U}_i} \{h_i\}, \quad i = 1, 2
\end{aligned} \tag{3}$$

$\boldsymbol{\lambda}_i$ is the time-dependent co-states for both players concatenated into one. The costates connect PMP and HJI via $\boldsymbol{\lambda}_i = \nabla_{\boldsymbol{x}_i} \nu_i$. Note that the solutions to Eq. (3) are unique to the initial states.

**Learning Values of General-sum Differential Games.** Directly learning values of a zero-sum differential game in a self-supervised fashion was first explored in [17]. In this approach, we directly fit a neural network to satisfy the governing HJI PDE. Let $\hat{\nu}_i(\cdot,\cdot,\cdot) : \mathcal{X}_i \times \mathcal{X}_{-i} \times [0,T] \to \mathbb{R}$ be a neural network parameterized by $\theta$ that approximates $\nu_i$. With $\mathcal{D} = \left\{ \left( \boldsymbol{x}_1^{(k)}, \boldsymbol{x}_2^{(k)}, t^{(k)} \right) \right\}_{k=1}^{K}$ representing the uniform samples in $\mathcal{X}_i \times \mathcal{X}_{-i} \times [0,T]$, the loss function that guides the learning of the general-sum value is:

$$\begin{aligned}
L_1(\hat{\nu_1}, \hat{\nu_2}; \theta) := \sum_{k=1}^{K} \sum_{i=1}^{2} \left| H\left( \hat{\nu}_i^{(k)}, \nabla_{\mathbf{x}_i} \hat{\nu}_i^{(k)}, \mathbf{x}_i^{(k)}, t^{(k)} \right) \right| \\
+ C_1 \left| B\left( \hat{\nu}_i^{(k)}, \mathbf{x}_i^{(k)} \right) \right|
\end{aligned} \tag{4}$$

where, $\hat{\nu}_i^{(k)}$ is the output of the neural network, $\hat{\nu}_i^{(k)} = \hat{\nu}_i \left( \boldsymbol{x}_i^{(k)}, \boldsymbol{x}_{-i}^{(k)}, t^{(k)} \right)$. $C_1$ balances the HJI PDE loss ($H$) and the boundary loss ($B$).

Note that at each training iteration, we compute the control policies for each player by maximizing their equilibrial Hamiltonian. We refer the readers to [10] for more details on challenges and different methods of learning values using this approach. For the purpose of this paper, we only consider the self-supervised learning method to learn the value function.

## V. METHODOLOGY

In this section, we present the algorithms for generating open-loop trajectories via solving BVP and training the value network using self-supervised learning method.

### A. BVP Solver

In this subsection, we discuss about generating open-loop trajectories. We use scipy's `solve_BVP` function to solve Eq. (3). To successfully solve the BVP, we first compute the analytical expressions for the quantities in Eq. (3). These are (1) the augmented dynamics which contains the state dynamics ($\dot{\boldsymbol{x}}$) and the co-state dynamics ($\dot{\boldsymbol{\lambda}}$), and (2) the boundary conditions. Note we also need a sub-routine to compute the control policies for each of the agents. Furthermore, convergence requires good guesses of the solution along the trajectory. To do so, given an initial state, we solve the state and co-state trajectories by solving their respective ODEs obtained from PMP.

### B. Multi-Agent RL using Nash Q-Learning

We also formulate the 2-regions case as a Reinforcement Learning problem and train the sub-swarms using Nash Q-Learning. The Nash Q-function for a learning agent $i$ is defined on the state (s) of the system, and its action (a) along with the actions of the other players, whereas in the Q-learning algorithm, the Q-function is only a function of the agent. Furthermore, in Q-learning, agents update their Q-values by accounting for future values as a result of maximizing their Q-values. In contrast, in Nash Q-learning, agents update their Q-values by following a Nash equilibrium strategy. More formally, the update is as follows:

$$\begin{aligned}
Q_{t+1}^i(s, a^1, \ldots, a^n) = (1 - \eta_t) Q_t^i(s, a^1, \ldots, a^n) + \\
\eta [r_t^i + \beta Nash Q_t^i(s')]
\end{aligned} \tag{5}$$

where, $r$ is the instantaneous reward, $\eta$ is the learning rate, and $\beta$ is the discount rate. $Nash$ is the operator of choice that computes the Nash equilibrium of the stage game at state $s'$. In this work, we use *support enumeration* as a nash operator. The Nash-DQN algorithm follows the vanilla Nash Q-Learning [20] with a Deep Q-network approximating the Q-values instead of the tabular method. The algorithm is identical to Deep Q-learning [23] with the policy replaced by the nash equilibrium.

Unlike the vanilla Nash Q-learning, we use two neural networks that approximate the Q-function of the sub-swarms in the 2-regions case. Each sub-swarm has an action space of 4 – (0, 0), (0, 1), (1, 0), (1, 1). Note that each agent must also know the Q-values of the other agent to compute the Nash Equilibrium. However, this information is not available, so each agent makes a conjecture about the other agent's Q-values. We achieve this by defining the Q-function on the joint action space of both sub-swarms. The result is a $4 \times 4$ matrix of Q-values for each sub-swarm. At each learning stage, each sub-swarm solves a bi-matrix game as follows: where, $a$ is the q-value for agent $i$, and $b$ is the q-value for agent $-i$ as conjectured by agent $i$. Methods such as *Lemke-Howson* [24] or *support enumeration* can be used to compute the Nash Equilibria for the bi-matrix game. Note that we use *NashPy*'s implementation of *support enumeration* [25]. Note that table I gets significantly larger when we consider higher dimensional case studies. Solving the bi-matrix game becomes a challenge for these cases. As a result, we only compare the performance of Nash DQN with PINN for the case study with 2 regions.

TABLE I: Bi-matrix stage game for Nash Q-Learning.

| | | Agent $-i$ | | | |
|---|---|---|---|---|---|
| | | (0, 0) | (0, 1) | (1, 0) | (1, 1) |
| Agent $i$ | (0, 0) | $a_1, b_1$ | $a_2, b_2$ | $a_3, b_3$ | $a_4, b_4$ |
| | (0, 1) | $a_5, b_5$ | $a_6, b_6$ | $a_7, b_7$ | $a_8, b_8$ |
| | (1, 0) | $a_9, b_9$ | $a_{10}, b_{10}$ | $a_{11}, b_{11}$ | $a_{12}, b_{12}$ |
| | (1, 1) | $a_{13}, b_{13}$ | $a_{14}, b_{14}$ | $a_{15}, b_{15}$ | $a_{16}, b_{16}$ |

### C. Self-supervised learning

In this subsection, we discuss the method of self-supervised learning algorithm using PINN to learn the value function. We implement curriculum learning by first learning the value at the final time and gradually increasing the time horizon starting from the end time. We provide a simplified

algorithm for training the value network. In our experiment, we use ADAM to optimize the neural network parameter $(\theta)$ with a decaying step-size (learning rate) $\gamma \in [2e-5, 1e-6]$. We use a neural network with 3 hidden layers containing 64 neurons each with `tanh` activation function, with the final layer being linear. For the higher dimensional case study we increase the hidden layer to 5 and the neurons to 128 at each layer. We use NVIDIA A100 with 2 GPUs for all the training in this paper. The input dimension to the neural network depends on the number of regions in the graph. We further reduced the dimension of the system using the fact that the densities of each sub-swarm sum to 1. Hence, for the case with 2 regions, the value function is a 3-dimensional function instead of 5, including time.

**Algorithm 1:** Self-supervised Learning

**input :** $T$, $num\_epoch$, $pretrain\_iters$, $k$, $n$
**output:** $V_\theta$

initialize neural network $V_\theta$;
add $k$ samples of $(\boldsymbol{x}_1, \boldsymbol{x}_2, 0)$ to $\mathcal{D}$;
pre-train $V_\theta$ at the boundary $(t = 0)$ for $pretrain\_iters$ iterations;
set $itr = 0$;
**while** $itr \leq num\_epoch$ **do**
  add $k$ samples of $t \sim \mathcal{U}(0, T(itr)/num\_epoch)$ to $\mathcal{T}$;
  add $k$ samples of $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ to $\mathcal{X}$;
  $\mathcal{D} \leftarrow (\mathcal{X}, \mathcal{T})$;
  append $n$ samples at the boundary $(t = 0)$ to $\mathcal{D}$;
  compute total loss $(L_1)$ using PDE loss $(H)$, and boundary loss $(B)$;
  update $\theta \leftarrow \theta + \gamma \nabla_\theta L_1$
**end**

## VI. CASE STUDIES

We test our method in three graphs representing low- and high-dimensional systems. We consider two sub-swarms with the same control capabilities interacting in an environment containing $M$ regions $(|\mathcal{V}| = M)$. We first consider $M = 2$ (as a toy case) and $M = 4$ regions with the former represented by a bidirected graph and the latter represented by a directed graph as shown in Fig. 1. We then consider a high dimensional case (21 D), with $M = 10$ regions, to show that the proposed method is easily scalable. The state of each sub-swarm is denoted by $\boldsymbol{x}_i \in \mathbb{R}^M$, with $M$ being the number of regions. The goal of each sub-swarm is to achieve a higher proportion in any of the regions (vertices in the graph). The terminal payoff to sub-swarm $i$ is $g(\boldsymbol{x}_i, \boldsymbol{x}_{-i})$ is a Boltzmann operator $(S_\alpha)$ defined on the element-wise difference between the states of the two sub-swarms. We use the Boltzmann operator instead of the $\max$ operator to make the function differentiable and continuous. The following represents the terminal payoff to the sub-swarm group $i = 1$. To compute the terminal payoff for sub-swarm group $i = 2$, simply reverse the order of difference.

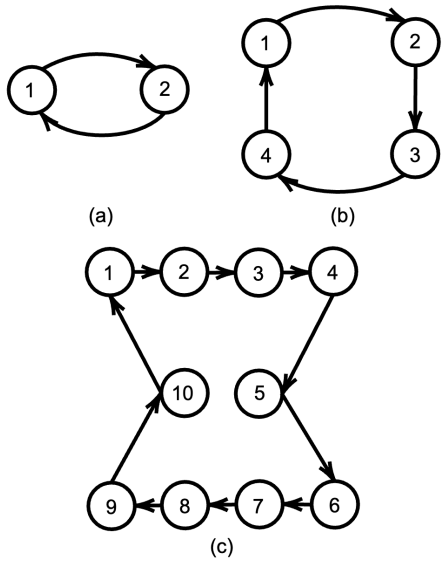


Fig. 1: (a) Directed graphs with 2, (b) 4, and (c) 10 regions. The maximum transition rate (control bounds) for all the edges is 1, and the minimum rate is 0.

$$g(\boldsymbol{x}_1, \boldsymbol{x}_2) = S_\alpha(\boldsymbol{x}_1 - \boldsymbol{x}_2) = \frac{\sum_{j=1}^M (\boldsymbol{x}_1 - \boldsymbol{x}_2)_j \, e^{\alpha(\boldsymbol{x}_1 - \boldsymbol{x}_2)_j}}{\sum_{j=1}^M e^{\alpha(\boldsymbol{x}_1 - \boldsymbol{x}_2)_j}} \tag{6}$$

where $\alpha$ is the temperature parameter and as $\alpha \to \infty$, it behaves like the $\max$ operator. We set $\alpha = 1$ for the experiments in this paper (unless mentioned otherwise). The gradient of $S_\alpha(\mathbf{y})$, $\mathbf{y} \in \mathbb{R}^M$ resembles that of a `softmax`:

$$\nabla_{\mathbf{y}} S_\alpha(y_1, \ldots, y_M) = \frac{e^\alpha y_i}{\sum_{j=1}^M e^{\alpha y_j}} \Big[1 + \alpha(y_i - S_\alpha(y_1, \ldots, y_M))\Big] \tag{7}$$

### A. Sub-swarms in 2 regions

We first solve a simple case with only 2 regions and identical homogeneous sub-swarms. We compare the open-loop policy obtained by solving the BVP using Eq. (3) with the policy obtained from the neural network.

**Training.** For 2-dimensional case, we sample 65k data and pretrain at the boundary for 10k iterations. We then implement curriculum learning by slowly increasing the time horizon and train for another 110k iterations. We also ensure that there are at least 10k data points at the boundary after the pretrain stage.

**Results.** Using the learned value network, we generate the closed-loop trajectories for a given initial condition representing the density of swarm groups in region 1. The results show that each swarm group will occupy one separate region at the end of the game.

We also observe that for some initial states, the value network and the BVP solver result in different equilibria. Since the problem is symmetric, it is possible that the swarms can occupy any one of two regions to guarantee a maximum

payoff as long as they pick separate regions. We show one example of such a case in Fig. 2.

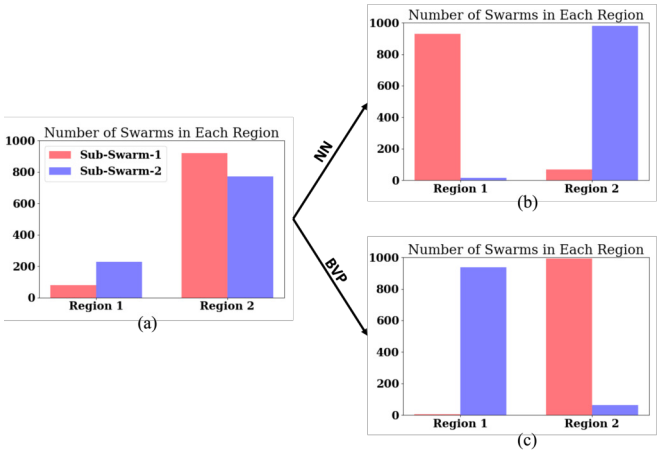


Fig. 2: A case demonstrating possible equilibria for the 2 regions case. (a) is the initial condition of the sub-swarms. (b) and (c) are the final distributions of the sub-swarms in the environment obtained from the value network and the BVP solver respectively. Multiple equilibria exist due to the symmetric nature of the game.

### *B. Sub-swarms in 4 regions*

Next, we consider an environment with 4 regions as shown in Fig. 1(c). Number of swarm groups is 2 with same transition rates across all edges.

**Training.** For the 4 regions case, we increase the training iterations to 200k with 20k pretrain steps. In addition to curriculum learning, we also increase the data points gradually starting from 30k in the beginning and adding 10k data points after every 10k iterations after the pretrain stage.

**Results.** As in the previous case study, the sub-swarms eventually find one region each where they dominate. Fig. 3 shows the density at the final time for four different initial conditions. All cases lead to the scenario where sub-swarms maintain dominance in one region.

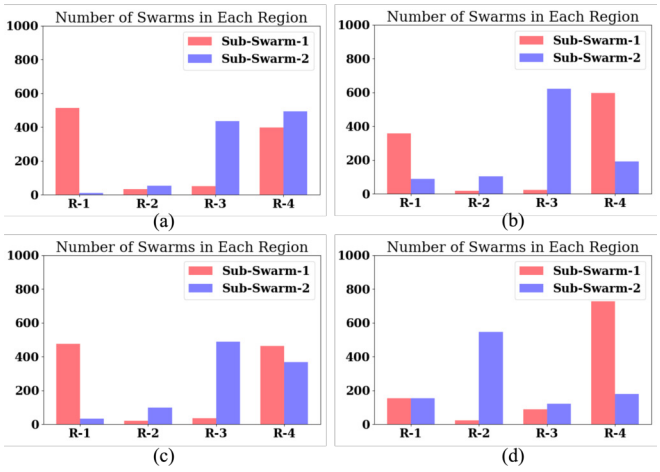


Fig. 3: (a-d) Represents the density distribution for two sub-swarms at the final time for four different initial conditions.

We observe that for some initial states, the final distributions are not optimal (see Fig. 3(a) and (c)). Sub-swarm-2 in Fig. 3(a) could get a higher payoff by aggregating in region 3 instead of two regions (3 and 4). This behavior can be attributed to the temperature parameter in the payoff function. Since hardening the temperature parameter led to difficulty in the convergence of BVP, we tested the effect of the temperature parameter using the value network. When $\alpha = 20$, results show that the sub-swarms proceed to aggregate in one region resulting in the desired behavior. Fig. 4 shows the final distributions of the sub-swarms for the same four initial conditions as in Fig. 3.

We compare the final payoffs across the learning algorithms for both 2 and 4 regions case studies. The results are reported in table II. A negative number means that the BVP solution resulted in a higher final payoff than the respective learning method. Final payoffs using PINN were close to the that obtained from the BVP solution whereas Nash DQN resulted in a significantly lower payoff on average.

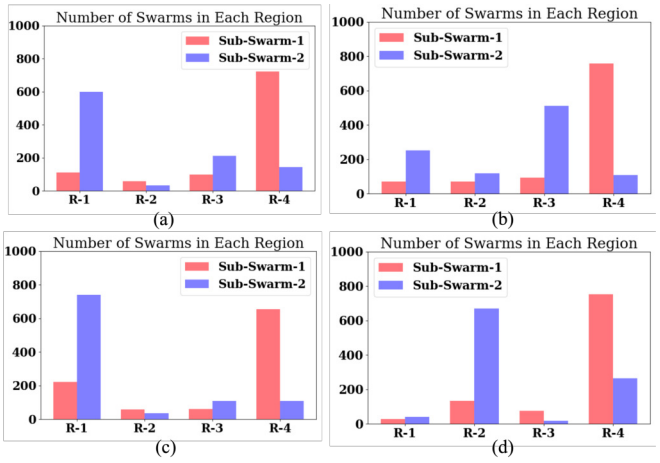


Fig. 4: Final distributions of the sub-swarms when the temperature parameter $\alpha = 20$. The sub-swarms attempt to aggregate in one region, which is desirable.

TABLE II: Comparison of final payoffs from solving the interaction using physics-informed neural network and Nash-DQN with BVP solver. $\overline{V_1}$ denotes mean final payoff to sub-swarm 1.

| Environment | $\overline{V_1}$ | $\overline{V_2}$ | Algorithm | $\overline{\hat{V}_1 - V_1}$ | $\overline{\hat{V}_2 - V_2}$ |
|---|---|---|---|---|---|
| Case-1 (2 regions) | 0.697 | 0.697 | PINN | **-0.011** | **-0.012** |
| | | | Nash DQN | -0.162 | -0.281 |
| Case-2 (4 regions) | 0.150 | 0.147 | PINN | **0.0001** | **-0.002** |

### *C. High Dimensional Case - 10 regions*

Finally, we present a higher dimensional case study for the environment shown in Fig. 1(c). There are 10 regions connected by a directed graph with 10 edges. Each sub-swarm group has to pick a 10-dimensional action at each time step. The action space of each sub-swarm is $2^{10}$, which

makes the problem intractable to solve in the case of Nash Q-learning and BVP.

**Training.** We slightly increase the neural network size to 5 hidden layers containing 128 neurons each. We keep other hyper-parameters the same as those in the 4 regions case.

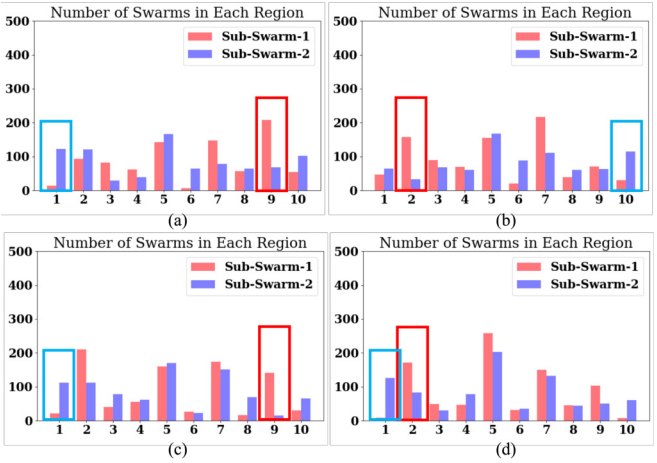


Fig. 5: Density distribution of two sub-swarms at the final time in the environment with 10 regions. Similar to the 4 regions case, the sub-swarms are able to find a region where they dominate. The regions where the sub-swarms dominate are bounded with a box.

**Results.** The final distributions of the sub-swarms across the 10 regions for 4 different initial conditions are shown in Fig. 5. From the plot, we see that there exists at least one region for both sub-swarms where their proportions are higher compared to the other sub-swarm. We do not observe a pronounced difference in densities of the sub-swarms as in the case of 4 and 2 regions. A possible explanation for not achieving this behavior could be an insufficient time horizon. It is possible that the time horizon of 2.5 seconds is not enough for sub-swarms to be able to aggregate in one region.

## VII. CONCLUSIONS

In this work, we demonstrated the efficacy of physics-informed neural networks in solving general sum games between swarms. Unlike conventional solvers that suffer from the CoD as the state dimension increases, PINNs can easily scale with the state dimension. However, an open question still exists regarding the validation of the policies due to the inherent black-box nature of the neural network, particularly for the higher dimensional cases where we lack an analytical solution. Future works will explore applications to real-life inspired problems such as network communications. Another possible direction is the introduction of information asymmetry. In the examples discussed in the paper, an implied assumption is made about the information availability. Strategies can get complicated with asymmetric information in the system. A natural extension of the present work is solving incomplete information games on swarms.

## REFERENCES


[1] M. Schranz, M. Umlauft, M. Sende, and W. Elmenreich, "Swarm robotic behaviors and current applications," *Frontiers in Robotics and AI*, p. 36, 2020.

[2] S. Berman, A. Halász, M. A. Hsieh, and V. Kumar, "Optimized stochastic policies for task allocation in swarms of robots," *IEEE transactions on robotics*, vol. 25, no. 4, pp. 927–937, 2009.

[3] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar, "Towards a swarm of agile micro quadrotors," *Autonomous Robots*, vol. 35, no. 4, pp. 287–300, 2013.

[4] W. Liu, A. F. Winfield, J. Sa, J. Chen, and L. Dou, "Towards energy optimization: Emergent task allocation in a swarm of foraging robots," *Adaptive behavior*, vol. 15, no. 3, pp. 289–305, 2007.

[5] X. Ai, V. Srinivasan, and C.-K. Tham, "Optimality and complexity of pure nash equilibria in the coverage game," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 7, pp. 1170–1182, 2008.

[6] E. Paraskevas, D. Maity, and J. S. Baras, "Distributed energy-aware mobile sensor coverage: A game theoretic approach," in *2016 American Control Conference (ACC)*, pp. 6259–6264, IEEE, 2016.

[7] K. Elamvazhuthi and S. Berman, "Mean-field models in swarm robotics: A survey," *Bioinspiration & Biomimetics*, vol. 15, no. 1, p. 015001, 2019.

[8] C. Sinigaglia, A. Manzoni, and F. Braghin, "Density control of large-scale particles swarm through pde-constrained optimization," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3530–3549, 2022.

[9] K. Tuyls, K. Tumer, and G. Weiss, "Multiagent learning," *Multiagent Systems*, pp. 423–475, 2013.

[10] L. Zhang, M. Ghimire, W. Zhang, Z. Xu, and Y. Ren, "Approximating discontinuous nash equilibrial values of two-player general-sum differential games," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3022–3028, 2023.

[11] M. G. Crandall and P.-L. Lions, "Viscosity solutions of hamilton-jacobi equations," *Transactions of the American mathematical society*, vol. 277, no. 1, pp. 1–42, 1983.

[12] I. M. Mitchell and C. J. Tomlin, "Overapproximating reachable sets by hamilton-jacobi projections," *journal of Scientific Computing*, vol. 19, pp. 323–346, 2003.

[13] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," *Advances in neural information processing systems*, vol. 30, 2017.

[14] K. Elamvazhuthi, Z. Kakish, A. Shirsat, and S. Berman, "Controllability and stabilization for herding a robotic swarm using a leader: A mean-field approach," *IEEE Transactions on Robotics*, vol. 37, no. 2, pp. 418–432, 2021.

[15] D. Milutinović and P. Lima, "Modeling and optimal centralized control of a large-size robotic population," *IEEE Transactions on Robotics*, vol. 22, no. 6, pp. 1280–1285, 2006.

[16] F. Djeumou, Z. Xu, M. Cubuktepe, and U. Topcu, "Probabilistic control of heterogeneous swarms subject to graph temporal logic specifications: A decentralized and scalable approach," *IEEE Transactions on Automatic Control*, vol. 68, no. 4, pp. 2245–2260, 2022.

[17] S. Bansal and C. Tomlin, "DeepReach: A deep learning approach to high-dimensional reachability," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[18] D. P. Bertsekas, "Rollout algorithms for discrete optimization: A survey,"

[19] D. Bertsekas, "Multiagent reinforcement learning: Rollout and policy iteration," *IEEE/CAA Journal of Automatica Sinica*, vol. 8, no. 2, pp. 249–272, 2021.

[20] J. Hu and M. P. Wellman, "Nash q-learning for general-sum stochastic games," *Journal of machine learning research*, vol. 4, no. Nov, pp. 1039–1069, 2003.

[21] P. Casgrain, B. Ning, and S. Jaimungal, "Deep q-learning for nash equilibria: Nash-dqn," *Applied Mathematical Finance*, vol. 29, no. 1, pp. 62–78, 2022.

[22] A. W. Starr and Y.-C. Ho, "Nonzero-sum differential games," *Journal of optimization theory and applications*, vol. 3, pp. 184–206, 1969.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[24] C. E. Lemke and J. T. Howson, Jr, "Equilibrium points of bimatrix games," *Journal of the Society for industrial and Applied Mathematics*, vol. 12, no. 2, pp. 413–423, 1964.

[25] V. Knight, katiemcgoldrick, M. Panayides, Y. Wang, A. S. Gaba, tokheim, I. F. Jr., O. Konovalov, J. Campbell, N. Glynatsi, P. Rivière, R. Baldevia, R. Szeto, arwheel, newaijj, and volume-on max, "drvinceknight/nashpy: v0.0.40," Aug. 2023.