2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS) | 979-8-3503-3744-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/ACSOS58161.2023.00021

Field-informed Reinforcement Learning of Collective Tasks with Graph Neural Networks

Gianluca Aguzzi
Alma Mater Studiorum — Università di Bologna
Cesena, Italy
gianluca.aguzzi@unibo.it

Mirko Viroli
Alma Mater Studiorum — Università di Bologna
Cesena, Italy
mirko.viroli@unibo.it

Lukas Esterle
Aarhus University
Aarhus, Denmark
lukas.esterle@ece.au.dk

***Abstract*—Coordinating a multi-agent system of intelligent situated agents is a traditional research problem, impacted by the challenges posed by the very notion of distributed intelligence. These problems arise from agents acquiring information locally, sharing their knowledge, and acting accordingly in their environment to achieve a common, global goal. These issues are even more evident in large-scale collective adaptive systems, where agent interactions are necessarily proximity-based, thus making the emergence of controlled global collective behaviour harder.**

In this context, two main approaches have been proposed for creating distributed controllers out of macro-level task/goal descriptions: *manual design*, in which programmers build the controllers directly, and *automatic design*, which involves synthesizing programs using machine learning methods. In this paper, we consider a new *hybrid* approach called *Field-Informed reinforcement learning* (FIRL). We utilise manually designed *computational fields* (globally distributed data structures) to manage global agent coordination. Then, using *Deep Q-learning* in combination with *Graph Neural Networks* we enable the agents to learn the necessary local behaviour automatically to solve collective tasks, relying on those fields through local perception. We demonstrate the effectiveness of this new approach in simulated use cases where tracking and covering tasks for swarm robotics are successfully solved.

***Index Terms*—Aggregate Computing, Graph Neural Networks, Cyber-Physical Swarms, Many Agent Reinforcement Learning.**

I. Introduction

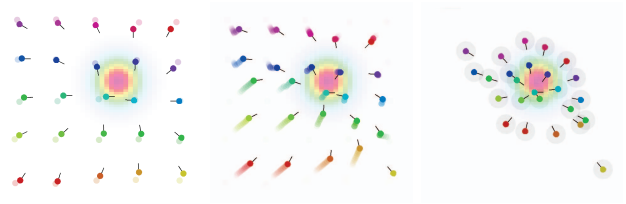
The coordination of a group of autonomous agents that can perceive and act in their environment is a fundamental problem in artificial intelligence. Such agents need to cooperate and communicate in order to achieve a common goal, while dealing with the challenges of distributed and situated intelligence. These challenges include the limited and local nature of the information available to each agent and the emergence of global behaviour from local interactions.

One domain where these challenges are particularly evident is Cyber-Physical Swarms (CPSs, or *swarm-like systems*), where a large number of agents interact with each other and their environment based on spatial proximity. Examples of CPSs include swarm robotics [1], smart cities [2], sensor networks [3], and social systems [4]. In these systems, agents need to adapt to dynamic and uncertain situations, while ensuring the achievement of global objectives that may not be directly observable or measurable by individual agents.

A key question in designing swarm-like systems is how to create distributed controllers for the agents that enable them to perform complex *collective* tasks. Two main approaches have been proposed for this problem: manual design and automatic design. In manual design, programmers build the controllers directly, using domain knowledge and programming languages or frameworks that support distributed computation and communication like macro-programming approaches [5]. In automatic design, machine learning methods like multi-agent reinforcement learning (MARL) and evolutionary algorithms are used to synthesize programs or policies for the agents from the high-level task or goal descriptions. Both approaches have advantages and disadvantages. Manual design allows programmers to specify desired properties for the system declaratively, however, it can be tedious and error-prone. Automatic design can overcome these limitations by learning from data and experience. However, automatic design can also suffer from several challenges, such as learning the right representation for agent communication.

In this paper, we propose a novel hybrid approach that combines manual and automatic design of distributed controllers. This follows a novel trend in which high-level declarative programming languages are mixed with machine learning techniques in order to synthesise robust collective controllers [6], [7], [8]. Specifically, we propose a solution called Field-Informed Reinforcement Learning (FIRL) that utilises aggregate computing [9] together with a graph neural network (GNN) [10] in combination with a reinforcement learning approach, namely Deep Q-Learning (DQN) [11]. Here the GNN is trained on field-information from aggregate computing and provides so-called node embeddings for each agent, serving as input for the DQN. The DQN provides then the appropriate actions for the agent to achieve their tasks. The main contribution of FIRL is the *definition of distributed controllers that are informed by collective knowledge that has been distilled during training, but that use only local information when deployed*. This way, FIRL can achieve a balance between manual design and automatic design, combining the benefits of both approaches while mitigating their drawbacks. Moreover, the learned policies have the potential to scale with size and adapt to different network topologies due to the inherent nature of GNNs and aggregate computations. FIRL can also be seen as a way of bridging the gap between symbolic and sub-symbolic AI methods, by integrating declarative programming with deep learning. Conceptually, FIRL

979-8-3503-3744-0/23/$31.00 ©2023 IEEE
DOI 10.1109/ACSOS58161.2023.00021



(a) Start (b) After 50 steps (c) After 200 steps

Fig. 1: Agents (coloured dots) are deployed in an area and have to coordinate to cover the phenomenon. The phenomenon can have varying areas of importance. Over time, the phenomenon will be covered sufficiently without any central controller.

leverages the ideas of *behaviour implicit communication* [12] [13], whereby intelligent agents (here trained by DQN) achieve collective goals by learning how to use signs they left in the environment (here made of fields): much like ants collectively rely on pheromones they produce [14].

We employ this approach in a swarm-like system setting where agents are tasked to cover phenomena detected in their environment. Over time, the agents have to converge over each phenomenon in order to cover it appropriately as illustrated in Figure 1.

The remainder of this paper is structured as follows. First, we introduce the relevant background and problem formulation in Section II. Afterwards, we introduce our approach in Section III. Section IV outlines the performed experiments and discusses the obtained results. Finally, we will present our conclusions in Section V.

II. Background and Motivation

A. Swarm systems

This article studies intelligent *collective behaviours* within the context of *large-scale* distributed systems. Specifically, we focus on *Cyber-Physical Swarms* or *swarm-like systems* consisting of *devices* governed by autonomous software *agents* and equipped with *sensors* and *actuator* able to interact with the real world. Each of these agents can interact with its direct neighbours, either based on physical (e.g. communication range) or logical distance (e.g. 1-hop neighbourhood).

Each agent executes a *local* control loop. In each iteration of this loop, the agent can access the information available within its own *context*. This context is comprised of information acquired by the agent directly through its sensors and the information received from its neighbours. At the end of each control loop iteration, the agent can send messages to its neighbourhood. This message may contain raw sensor data or aggregated information over time or from other neighbours. Examples of such large-scale systems can be a network of robots (e.g., swarm robotics), camera and IoT networks, or a network of mobile phones. Our goal is to find a *homogeneous* distributed controller π, namely the same controller for all agents of the system: starting from *only* local configurations, it leads the system to achieve a certain *collective* requirement through *cooperation*, such as spatial area coverage, phenomena tracking, and robot aggregation [15]. The homogeneity of the controller is a key requirement to ensure the *scalability* of the approach, as it allows us to avoid the need for a centralised controller that would be a bottleneck for the system, and it is the typical choice in swarm-like systems [1], [16], [17], [18].

B. Field-based Coordination

The field-based coordination approaches utilise a concept of *computational fields* (or simply *fields*), which are *distributed* data structures that associate each location with a computational value that evolves over time. *Aggregate computing* [9], a modern field-coordination approach, is rooted in earlier work on *artificial potential fields* [19] and *co-fields* [20]. This *macro-programming* [5] paradigm can be realised to devise collective and self-organizing behaviour through a composition of functions operating on fields. The fields map a set of individual agents to computational values, allowing them to associate what they sense, process, and actuate. Fields are computed locally but subject to a global viewpoint, enabling emergent collective behaviour through the interplay of the *system model* (i.e., structure and dynamics of the systems) and the *programming model* (i.e., how collective behaviours are expressed).

On the one hand, the system model is structurally similar to *swarm-like systems*, as it is composed of a set of *agents* that interact with each other through a *network* of *neighbourhood relations*. The dynamics of each agent follow a local control loop called *round* that comprises the following steps:

1) *Sense*: the agent acquires information from the environment and collects the messages of its neighbourhood, building a *context*;
2) *Compute*: the agent executes a *program* that computes an *export*, that is a set of values to be sent to the neighbourhood;
3) *Act*: the agent sends the export to its neighbourhood and updates its internal state acting on the environment.

The proactive and iterative execution of the round by each agent leads to the emergence of collective behaviour specified by the program.

On the other hand, the programming model is ruled by the *field calculus* [21], [22], a core language that allows the expression of collective behaviours through the composition of *functions* operating on fields. Field calculus includes main operators for expressing *spatio-temporal* computations that allow for *i)* the progression of values over time, achieved by transforming a field computed in a previous round into a new field; *ii)* the exchange of data with neighbouring fields (typically referred to as `nbr`), where received data is accessed by adjacent fields; and *iii)* the conditional division of computation into distinct domains of collective computation. For further elaboration on the actual field calculus, please refer to the work of [23].

On these minimal operators, it is then possible to build *self-organizing* coordination blocks. One of the founding blocks is the *gradient* essential for *information* flows [24]. This operator

generates a numerical field that represents the minimum distance from a source zone. In other words, it maps a Boolean field (where "true" indicates the presence of a source agent and "false" indicates its absence) to a distance field that indicates the proximity to the nearest source. In the ScaFi [25] implementation of field calculus is defined by function:

```
def gradient(source: Boolean): Double
```

Along the gradient, it is possible to cast information within the system, accumulating value during the field expansion. This pattern is called *gradient-cast* (or *G*):

```
def G[V](source: Boolean, value: V, acc: V => V): V
```

where `source` is a Boolean field indicating the presence of a source agent, `value` is the value to be cast, and `acc` is the accumulation function that is applied to the value during the cast. This is a fundamental building block for the coordination of swarm-like systems: in fact, the resulting system can be used to broadcast information, or route towards selected nodes.

C. Graph Neural Networks

GNN is a novel neural network model used to process graph-structured data with deep learning approaches. Let $G = (V, E)$ be a graph where $E \subseteq V \times V$ defines the neighbourhood relations for each participating node, and V identifies the nodes present in the graph. Each node $v \in V$ is associated with an observation (or feature set) f_v. For the sake of simplicity, we thereafter describe G_f as a graph that contains the feature set f_v for each node $v \in V$. Note that, when we refer to G_f and G_o we are referring to the same graph G but with different node features. Also, to access the feature set f_v of a node $v \in V$ we use the notation f_v or $G_f[v]$. Given f_v, the goal of a GNN is to learn the node embedding h_v for each node $v \in V$. The node embedding h_v describes the node in the network and summarises the geometric properties of the graph in this location, allowing for comparison of various nodes in the graph. In modern GNNs, the node embedding h_v is computed by aggregating information from the node's neighbours $\mathcal{N}_G(v)$, and then combining it with the node's current embedding h_v in a process called *message passing* [26]. The GNNs is partitioned into several message passing layer k where each of them is responsible for computing the node embedding $h_v^{(k)}$ for each node $v \in V$. Formally, a GNN can be defined by three phases:

$$m_{uv}^{(k)} = \psi^{(k)}\left(h_u^{(k-1)}, h_v^{(k-1)}, e_{uv}^{(k-1)}\right) \tag{1}$$

$$a_u^{(k)} = \bigoplus^{(k)}\left(\left\{m_{uv}^{(k)} : v \in \mathcal{N}_G(u)\right\}\right) \tag{2}$$

$$h_u^{(k)} = \phi^{(k)}\left(h_u^{(k-1)}, a_u^{(k)}\right) \tag{3}$$

where h_v^k is the embedding of node v within the k-th layer, $\mathcal{N}_G(v)$ is the set of neighbours of node v computed from E. h_v^0 is the initial embedding of node v, and it is usually set to the node's feature vector f_v. The differential part comes into play in the ψ and ϕ functions, which is usually a differentiable function such as a neural network. The ψ function is called *message function*, and it is responsible for computing the message $m_{uv}^{(k)}$ from node u to node v. The ϕ function is called *update function*, and it is responsible for updating the node embedding $h_v^{(k)}$ of node v. $\bigoplus$ instead is a function that aggregates the information from the neighbours of a node v and it could be a simple sum, max or sum of products and it should be permutation invariant. More complex aggregation functions are available [27]. Thereafter, we express the application of a GNN to a graph G_f as:

$$GNN(G_f) = \{h_v^{(k)} : v \in V, k \in \mathbb{N}\} \tag{4}$$

This formulation allows GNNs to effectively process and extract features from graph-structured data by iteratively aggregating and transforming information from the node's neighbours.

GNNs are used in several application areas such as social network analysis, chemistry, and physics. In this paper, we use GNNs to learn a local behaviour for each agent in a multi-agent system (more details in Section III).

D. Many-Agent Reinforcement Learning

Reinforcement learning (RL) has gained a lot of interest recently, thanks to its successful application in various scenarios, ranging from video games (such as Alpha Go [28] and Atari [29]) to chatbots (like ChatGPT [30]). In RL, an *agent* (i.e., a smart entity capable of making decisions) performs *actions* in an *environment* (i.e., everything outside the agent) according to a *policy*, to maximise long-term *reward signal*.

One interesting application of RL is when there are multiple learning agents involved. Such scenarios are referred to as multi-agent reinforcement learning (MARL) [31]. In particular, in this work, we consider *homogenous many-agent reinforcement learning (ManyRL)* [16], where the set of agents is large ($N \gg 2$) and each agent is *interchangeable* and *indistinguishable*. This research area is relevant in the context of large-scale systems where collective intelligence emerges from local and repeated interaction of simple entities, like in swarm robotics. In such many-agent scenarios, the implementation of fully decentralized learning is often unfeasible due to the large number of learning agents, which makes the system non-stationary and difficult to manage. Conversely, a centralized controller capable of coordinating the entire system may not be a viable solution due to scalability concerns. To address this challenge, a practical solution is the adoption of *centralized training and decentralized execution (CTDE)* approach. The idea is to learn a policy at simulation time when there is a collective view of the system, and then at runtime use that policy but only with local observations. The typical approach in such cases is based on actor-critic systems [32], [33], [34], [35], where the *actor* is the distributed policy (with only local information) and the critic is a neural network that takes the overall system state. Mean-field RL [17] is one of such concrete applications of CTDE where the interactions among

the population of agents are estimated by considering either the effect of a single agent and the average impact of the entire population or the influence of neighbouring agents. Some known approaches using mean-field reinforcement learning include Q-mean, which is an extension of Q-learning to mean-field settings [36], and actor-critic mean-field [37], which combines actor-critic algorithms with mean-field approximations. These approaches have shown promising results in various domains, such as multi-agent coordination and decentralised control, and are actively being researched and developed for further applications.

E. Problem formalisation

Given the homogeneity, large system scale, and the *locality* (i,e., each agent can only observe its neighbours), the problem can be modelled through the SwarMDP model [38]—an extension of the decentralized partially observable Markov decision processes (DecPOMDP) [39] model for swarm-like systems. A SwarMDP is characterised by a *swarming agent* ($\mathbb{A}$) and the dynamics of the environment ($\mathbb{E}$). Specifically, $\mathbb{A}$ is a tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{R}, \pi)$ where:

- $\mathcal{S}, \mathcal{O}, \mathcal{A}$ are the set of local states, observations (or features), and actions, respectively;
- $\mathcal{R} : \mathcal{S} \to \mathbb{R}$ is the reward function, which is influenced by the environment;
- $\pi : \mathcal{O} \to \mathcal{A}$ is the policy function, which maps the observations to the actions: it could be deterministic or stochastic.

Starting from this definition, the environment $\mathbb{E}$ is defined as a tuple $(\mathcal{P}, \mathbb{A}, \mathcal{T}, \xi)$, where:

- $\mathcal{P}$ is the total number of agents in the systems (the agent population), which is assumed to be fixed;
- $\mathbb{A}$ is the defined agent prototype that rules each agent $v \in P$;
- $\mathcal{T} : \mathcal{S}^P \times \mathcal{A}^P \times \mathcal{S}^P \to \mathbb{R}$ is the transition global function, which is influenced by the actions of the agents and returns a collective reward – this is typically not known by the swarming agents;
- $\xi : \mathcal{S}^{\mathcal{P}} \to \mathcal{O}^{\mathcal{P}}$ is the global observation model of the systems.

In swarMDP, the neighbourhood is not directly defined, but it is implicitly defined by the observation model ξ. In our specific case, the agents can only interact with 1-hop neighbours and are not directly influenced by other agent observations. We can therefore restrict the observation model as follows:

$$\xi(v) : \{s_j, j \in \mathcal{N}^v\} \to O \qquad \xi = \{\xi(v), v \in \mathcal{P}\}$$

where $\mathcal{N}^v$ is the set of neighbours of v. This model can be used then to express the evolution of the system in time. Specifically, starting from a global state $\mathcal{S}_t^P$, the next state $\mathcal{S}_{t+1}^P$ is defined as:

$$\mathcal{A}_t^P = \pi(\xi(S_t^P)) \qquad \mathcal{S}_{t+1}^P = \mathcal{T}(\mathcal{S}_t^P, \mathcal{A}_t^P)$$

Given a time t, the system can be also represented as a graph $G^t = V^t, E^t$, where E^t is built from $\mathcal{N}$. Each node is then decorated with the local observation perceived at the time t: $o_t^v \in O$. This graph can be used both to compute computational fields and, as done in previous work [40], [41], [42], can be the input for a GNN.

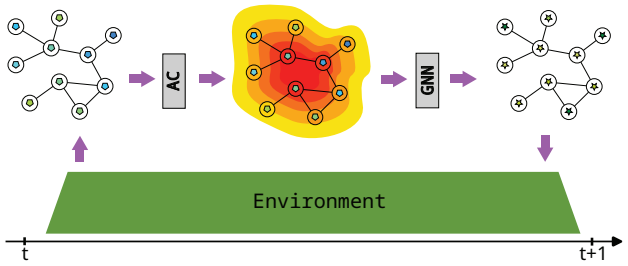


Fig. 2: High-level description of FIRL approach. For each time step t, a graph is constructed from the environment, and it will associate each node with a local feature o_t^v (hexagons in the picture). Using this feature, an aggregate program computes spatio-temporal information that enhances the feature set of each agent producing f_v, depicted as colours in the middle graph. Finally, utilizing the GNN, actions are computed for each agent in the system to be performed against the environment, enabling advancement in the simulations according to swarMDP rules.

F. Motivation

As mentioned above, our work falls in between automatic and manual approaches, specifically in the areas of field-based coordination and ManyRL with the use of GNNs. Compared to earlier works on field-based coordination, our approach builds on the concept of co-fields [20], where agents construct and exploit the field as a "digital sign" [12] to receive system-wide information and apply reasoning to this data. We present a subsequent approach where agent intelligence is synthesised through ManyRL and GNN is used to learn a local representation from the neighbourhood. By learning a smart policy directly in the environment, the agent becomes capable of adapting its behaviour to new situations.

The use of GNNs as part of a distributed controller has been explored in previous literature [42], [40], where it was shown that they could be used to break down the evaluation of local and distributed systems. However, in these works, communication was left entirely to the neural network, making the learning process potentially more complex and unstable. In our approach, the GNN is *informed* by computational fields that collect the necessary information to compute a certain task, limiting learning to only the specific task defined by a collective reward function. This will speed up the learning process and make it more stable.

III. Field-informed Reinforcement Learning

In this section, we discuss the components involved in our proposed solution (i.e., architecture) and how these components interact with each other to bring the system to perform the collective behaviour (i.e., dynamics). Finally, we will detail

the learning algorithm designed and used to synthesise the policy.

A. Architecture, fields and aggregate dynamics

The proposed solution, summarised in Figure 2, consists mainly of two parts, *i)* the aggregate program used to create part of the observation and *ii)* the policy π_{gnn} learned through GNN-based approach. Let Γ be the aggregate program that takes a graph G^t decorated by o_t^v, representing the participating agents and their neighbourhood relations at time t, as input. The evaluation of Γ produces a field value θ_t^v for each node v in G^t. From this field, we construct the feature vector f_v for each node v in G^t as follows: $f_v = (\theta_t^v, o_t^v)$. The policy π_{gnn} is then evaluated for each agent using f_v as input, producing an action a_t^v that will then modify the global state of the system. While the graph, containing the aggregate information, might appear as global knowledge, this is not the case as the information is *never* aggregated globally. The individual agents only combine information from their local neighbourhood. In fact, the program Γ is proactively executed at every agent, and the GNN can be locally evaluated using only neighbourhood information. We want to emphasise that, in this case, the GNNs must be 1-hop; otherwise, they could not have a local interpretation for each agent, according to our system model.

B. Learning algorithm

Since we consider swarm-like systems, which are an example of many-agent systems, the proposed approach follows a CTDE learning pattern, but differently from mean-field approaches and actor-critic solutions, we use a *value-based* approach combined with a GNN as a function approximator. Specifically, we leveraged the property of GNNs to have a dual interpretation, i.e., to function globally over the entire graph and locally only over the neighbourhood. Importantly, each agent only has local information from itself and its neighbourhood to utilise in the GNN. As value-based algorithm we relied on DQN [29] with two major modifications (see Algorithm 1):

1) experience replay stores experiences in the form of *graphs* decorated with features (e.g., observations, actions, rewards, etc.),
2) the neural network used to compute the Q function is based on a GNN with an multi-layer perceptron (MLP) downstream.

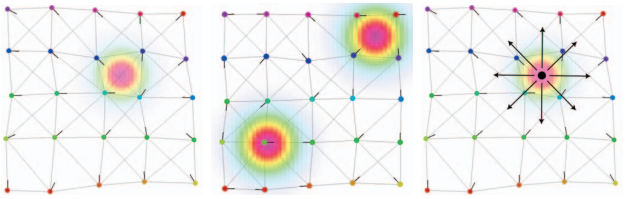
The first point is a natural extension because we work on graphs rather than simple values. This also influences how we create a batch of experiences to train the network. In fact, we sample a batch of graphs from the replay buffer, and then we merge them into a single graph, which is then used to train the network. This process is called *graph mini-batching* [43], [44] and its main purpose is to pass an entire batch of graphs to the same GNN for improved performance. For the second point, the use of GNNs allows us to define policies on a variable neighbourhood, which is essential in such systems as this can change due to the applied neighbourhood policy. It is known that GNNs have a certain ability to generalise to new structures and scale with different agents [45], [46]. Additionally, using the overall graph compared to local experiences makes learning more stable as it reduces the non-stationarity of the environment perceived by each node. This is because, even though the actions are produced using only local and neighbourhood information, during the learning phase, we have access to the internal graph, which will influence the policy through non-local information during the backpropagation.

Algorithm 1: Deep Q-Network (DQN) with GNN and Graph Replay Buffer executed by each agent

Input: Environment $\mathbb{E}$, graph replay buffer $\mathcal{D}$, target network θ^-, current network θ, exploration strategy ϵ
Output: Trained DQN model θ
Initialise $\mathcal{D}$ with random initial transitions;
Initialise θ with random weights;
Set $\theta^- \leftarrow \theta$;
while *not done* **do**
  Observe current graph observations G_o;
  if *random* $< \epsilon$ **then**
    select a random action a;
  else
    $G_q = Q(G_o, \theta)$;
    $a = \{v \in G_q | a_v \in argmax_{a_v} G_q[v](a_v)\}$;
  end
  Execute the collective action G_a in the environment $\mathbb{E}$ and observe a graph-level reward G_r and the next observation G'_o;
  Store transition (G_o, G_a, G_r, G'_o) in $\mathcal{D}$;
  Sample a batch of graph transitions $(G_o^i, G_a^i, G_r^i, G_o'^i)$ from $\mathcal{D}$ and merge them in $(G_o^b, G_a^b, G_r^b, G_o'^b)$;
  Compute the target Q-value for each node v in the graph G^b:
  $y_v = G_r^b[v] + \gamma * \max a' Q(G_o'^b[v], G_a^b[a']; \theta^-)$;
  Compute the current expected value for each node v in the graph G^b: $y_v^* = Q(G_o^b[v], G_a^b[v]; \theta)$;
  Update the current network weights using gradient descent: $\theta \leftarrow \theta - \alpha \nabla \theta \frac{1}{|G^b|}(y - y^*)^2$;
  Every C steps, update the target network weights: $\theta^- \leftarrow \theta$;
end

IV. Evaluation

To test the effectiveness of the proposed approach, we experiment with a case study related to swarm robotics, specifically, tracking and coverage of a spatio-temporal phenomenon—cf. tracking a wildfire or monitoring the water levels in a canal with multiple autonomous agents embodied in embedded devices (e.g., drones or IoT devices). The individual agents do not have any knowledge of the initial phenomenon itself (i.e., shape, size, location, velocity, and so on). Initially, we perform the training phase using a stationary phenomenon before expanding towards a moving phenomenon in the test



(a) (b) (c)

Fig. 3: Simulations of the case study scenario in Alchemist. The dots represent the agent, the circle area on represents the phenomenon to be monitored. Figure 3a represents the scenario used during training as well as during test. The others instead are only used in the test phase to evaluate the policy found.

phase. Finally, the phenomenon may have varying areas of interest, defined by an underlying distribution function. This underlying distribution is utilised in the feature set of each agent's observation and guides the agents to rally over the phenomenon. While we only use Guassian distributions but we can use any other distribution and shape. As a simulation environment, we use Alchemist [47], a simulator for multi-agent systems that allows us to simulate the swarm behaviour of the agents and the phenomenon of interest. For the GNN, we use the implementation provided by PyTorch Geometric [48], which is a library for deep learning on graphs built on top of PyTorch [49]. Finally, we use ScaFi [25] as the aggregate programming language to support our field-informed approach. The evaluation is performed in two stages, first the neural networks are trained in an explicit training phase before being extensively evaluated in the testing stage. [1]

A. Scenario

Figure 3 presents the three different types of scenarios utilised within the evaluation. The first type of experiments considers a single phenomenon at a static location (i.e., *Zone Fixed*), the second type of experiment considers two phenomena in two independent but static locations (i.e., *Two Zones*), and the third type of experiment considers a moving phenomenon (i.e., *Moving*). All phenomena are modelled as a Gaussian distribution. Importantly, only the left scenario illustrated in Figure 3a was used for training the neural networks. Furthermore, all three types of scenarios contain a set of $\mathcal{P} = 25$ agents placed in a 2D grid large 1000x1000 meters. Each agent can perceive the presence of the phenomenon of interest through an installed sensor ζ_v with $v \in \mathbb{V}$. (e.g., camera, temperature sensor, etc.) if it is within range. Additionally, each agent has a coverage range ω (fixed to 75 meters) that describes the area it can monitor. Each agent can only communicate directly with its own neighbourhood $\mathcal{N}$, which in this case depends on a $\mathbb{O}$ range fixed to 300 meters. Through this communication channel, agents can exchange information. Each agent moves following a certain action composed of two components (r, i) which respectively describe the angle and intensity of the movement (i.e., the velocity vector). Since we used a value-based approach, the action space $\mathcal{A}$ is discrete and composed of 18 possible angles and 3 possible intensities. In particular, the angles are quantized to 20 degrees, and for the velocities, we have selected [0, 5, 10] m/s.

For the aggregated information, each agent will produce a computation field with which they will try to approximate the direction of the phenomenon of interest. The program Γ in question is a simple application of block `G`, where the source is the maximum value of the neighbourhood. This can be expressed in ScaFi as follows:

```
val source = maxHood(nbr(sense(ζ))) == sense(ζ)
G(source, Point3D.Zero, _ + nbrVector())
```

where `maxHood` is a function that returns the maximum value of the neighbourhood, `nbr` is a function that a neighbourhood field of values (in this case, the sensor value ζ), `sense` is a function that returns the value of the sensor, and `nbrRange` is a function that returns an approximate direction for each agent in the neighbourhood. This value will then be fed into the π_{GNN} to compute the action to be performed.

B. Goal

The objective of this scenario is threefold:

1) maximise the number of agents within the phenomena;
2) minimise the number of agents without neighbours;
3) maximise the coverage of the system.

As we are modelling a reinforcement learning system, these three components must be encoded in a reward function that provides an estimate of the current action taken by a given agent. Formally, we define the reward of an agent being within the phenomenon as:

$$R_v^a = 1 \text{ if } \zeta_v > 0 \text{ else } 0 \tag{5}$$

Namely, an agent is considered within the phenomena as soon as the drone can sense the phenomenon. This will lead the system to prefer a configuration in which every agent is present within the phenomenon. The second element in the objective function ensures cohesion among the agents. This is important because if the system breaks into many scattered agents, the observability of the phenomenon is reduced, limiting the ability of the agents to move appropriately in the environment. In this case, the reward is defined as:

$$R_v^{\mathcal{N}} = 1 \text{ if } |\mathcal{N}| > 0 \text{ else } 0$$

Finally, to maximise the coverage, we define a reward function that favours the maximum distance between the agents equal to the coverage range ω. This will minimise multiple agents covering a common area. This means that the average distance will tend towards the one expressed by the viewing range of each agent:

$$R_v^C = 1 - \frac{d_{min}}{\omega}$$

[1]The simulations are publicly available at https://github.com/AggregateComputing/experiment-2023-acsos-field-informed-rl.

Where d_{min} describes and minimum distance of the agent to its neighbourhood. The final reward function R_v for an agent v is defined as:

$$R_v = (\frac{R_v^a + R_v^{\mathcal{N}} + R_v^C}{3}) - 1$$

Specifically, we decided to express the signal as a *regret* as it is a more general measure of the quality of the action taken by the agent.

C. Training Phase

Before we can evaluate our approach, the underlying neural networks have to be fine-tuned in a dedicated training phase. The training process for each neural network was divided into 100 episodes, each consisting of 200 steps, resulting in a total of 20,000 experiences. For each episode, the 25 agents are semi-randomly positioned on a grid (i.e., in a lattice layout with a random variation in their position) without knowing the correct position of the phenomenon, but that is fixed in the top right corner. The position of the phenomenon with an example of positioned agents can be seen in Figure 3a. The feature set used by the GNN created for each agent consists of the vector computed by the aggregated program and the value of the local sensor ζ. In this case, we chose to use an exponential epsilon decay, defined as: $\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) \cdot e^{-\lambda \cdot e}$. Where e is the current episode number. This leads to a high number of random actions at the beginning and gradually shifts towards exploitation in the later episodes. In our training process, we set $\epsilon_{min} = 0.02$, $\epsilon_{max} = 0.99$, and $\lambda = 0.1$. γ was set to 0.99, as we want to give more value to future returns, aiming to achieve good coverage by continuously tracking the phenomenon. The neural network structure used consists of a layer of SuperGAT [50] – a GNN based on attention mechanisms – and a layer of MLP. The hidden size was set to 256. As the reward function is defined as a regret, we decided to use the Huber loss function with $\delta = 1$. This function is used to penalise the agent if the action taken is too far from the optimal action. We use the RMSprop optimiser with a learning rate of 0.0001. Finally, we use a replay buffer of size 1000 to store the graph experiences and a batch size of 32.

D. Test phase

For the evaluation, we explore the previously discussed three different types of experiments. We generated 64 random scenarios for each type of experiment. Additionally, the placement of the agents was randomised as it has been done during training. For the first type, consider a single static phenomenon randomly placed in the environment. This is in contrast to the training where the phenomena were always placed in the same location. For the second type, we placed two distinct phenomena within the area. Their location is kept constant in all 64 experiments. As the training only contained a single phenomenon, this setup represents a challenge for the agents. Finally, the third type contained moving phenomena. In each scenario, the starting position as well as the direction of movement is randomly sampled from a uniform distribution. The movement is in a straight line with a constant speed of 5m/s within an unbounded environment. Examples of all three types of experiments are shown in Figure 3.

E. Baselines

We compare our FIRL approach against baseline approaches where the DQN utilises a MLP as well as an approach only relying on GNNs, without additional field information. In all approaches, the underlying neural network (i.e., the MLP and the GNN) are trained with a single, stationary phenomenon.

The MLP uses the same feature set as the GNN but applies it in the DQN but without leveraging the graph structure. Moreover, we increase the batch size to 512 and the replay buffer to 10000 since we record 25 agents' experiences for each step instead of one graph experience. The GNN alone, without using the field information, apply the position of agents and the local sensor value directly as input features within the DQN. These baselines are used to verify the effectiveness of the components used in the FIRL. Indeed the MLP baseline is used to verify the effectiveness of the GNN in the proposed approach, while the GNN baseline is used to verify the effectiveness of the field information in the proposed approach.

F. Metrics

We evaluate the performance of the different approaches by measuring the coverage of the phenomenon over time. The coverage is defined as the percentage of the phenomenon covered by the agents. Specifically, we can measure the coverage as the intersection over the union of the phenomenon and the agents' view range. Formally, we define the overall coverage for a certain time step as:

$$\Omega = \bigcup_{v \in V} \omega_v \quad C = \frac{|\Omega \cap \mathcal{P}|}{|\mathcal{P}|}$$

where $\mathcal{P}$ is the area of the phenomenon and Ω is the area covered by the agents. In the training phases, we measure the average coverage in each episode, and the total reward obtained by the agents at each episode of the simulation. We also measure the number of agents that are within the phenomenon at each step of the simulation. This will be a measure of how well the agents are tracking the phenomenon.

G. Discussion and Results

The results of the training process are summarized in Figure 4. In the charts, the line represents the average value of a metric of interest, while the shaded area represents its standard deviation. Specifically, we observe that the proposed version achieves higher coverage and total reward compared to other approaches. Interestingly, despite the global information available in GNNs without fields, they fail to converge to a good result like the one obtained with the field. This outcome was expected, as the computed field helps agents encode the necessary information to navigate towards the phenomenon. Furthermore, we note that GNN combined with DQN and graph replay buffer outperforms the simple MLP informed

field computation. This is because relying solely on MLP and basic deep learning leads to non-stationary and unstable learning, as evident from the wider confidence interval of the reward over training time.

Focusing now on the results of the test phase, highlighted in Figure 5, we observe that the field-informed version achieves higher coverage than the other approaches in all scenarios since it shows the ability of our solution to generalize to situations. We observe that the field-informed version successfully moves the agents closer to the target phenomenon, distributing them evenly without collapsing into a single central point. Figure 5 quantitatively presents the results across various previously described scenarios.

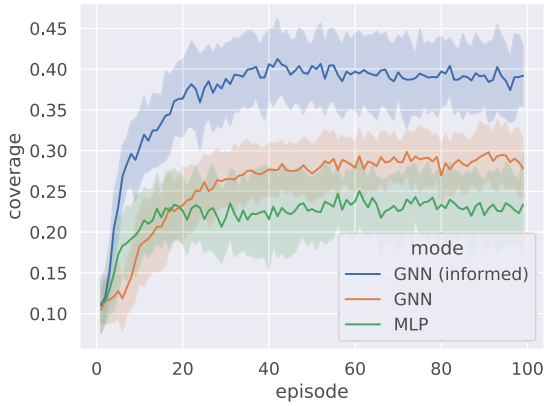
For all experiments, both GNN versions demonstrate the capability to transfer the learned experience to the test phase whereas the MLP version fails to generalize. It is worth noting that, in the *Zone Fixed* experiment, once the desired configuration is achieved in the static case, the agents cease to move, maintaining the found configuration. Interesting observations arise when we use scenarios different from the training phase. In the *Two Zones* experiment, we notice that our approach using field-information finds a better configuration than the simple GNN counterpart. It exhibits both higher overall coverage and manages to divide the system into two equally covered parts. Indeed, observing the Figure 6, we notice that the informed version maintains a balanced coverage between the two zones, with a difference of less than 5% between the two parts, maintaining a fair division of the phenomena. In contrast, the uninformed version also maintains a fair division but with significantly different coverage between the two parts, indicating a wrong placement among the agents in one of the zones. This is a consequence of the uninformed version's inability to encode the necessary information to divide the agents into two zones, therefore it is not able to generalise. Finally, the *Moving* experiment emphasizes how the informed version generates a more robust policy for new scenarios. Indeed, we observe that our approach using FIRL maintains higher coverage and a greater number of agents on the target phenomenon compared to the other two solutions. The uninformed GNN version, however, fails again to generalize its movement behaviour, as evidenced by the simulations where the agents, once reaching the target zone, stop moving due to tracking issues.

In conclusion, the results demonstrate how the proposed idea can generate more robust controllers. By guiding information flow in GNNs, we improve learning efficiency and alleviate the challenge of encoding relevant information. Nevertheless, we acknowledge the crucial role of GNNs. Our modified version of DQN, combined with GNNs, enables the discovery of robust behaviours in a few episodes, which is challenging to capture with MLPs combined with DQN, even if we use field information.

V. Conclusion and Future Work

In this paper, we have introduced a novel approach for constructing distributed controllers by leveraging aggregate computing to encode agent interactions, along with the combination of DQN and GNN for synthesizing distributed intelligence. The proposed *Field-Informed reinforcement learning* (FIRL) approach offers a promising solution to the challenges faced in coordinating multi-agent systems. By combining manual design and machine learning techniques, the approach enables agents to autonomously learn and adapt their behaviour while leveraging locally available information. The demonstrated success in the proposed case study in solving collective tasks underscores the potential impact of this approach in advancing the field of multi-agent systems and swarm robotics.

Future research could explore its application in diverse domains and evaluate its scalability and robustness in increasingly complex scenarios. In addition, we also plan to take the approach to modern actor-critical solutions, which are better suited to modern swarm robotics problems because of the continuous action space.



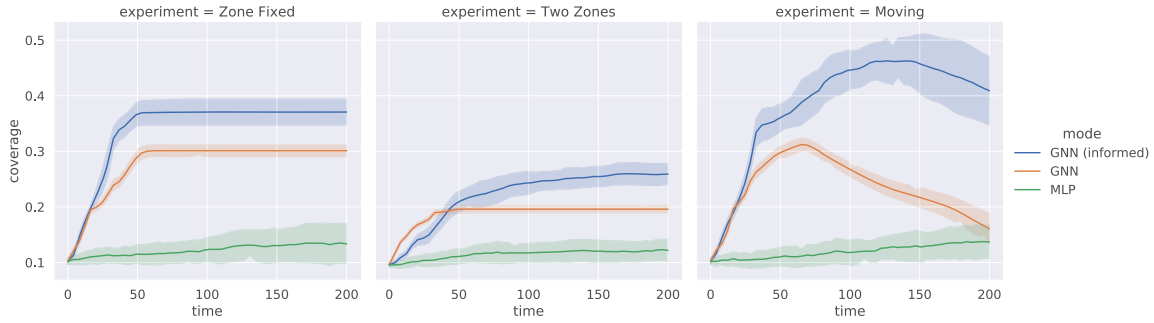
(a) Average coverage for each episode in training

(b) Reward during training

Fig. 4: Training results of FIRL. It can cover the phenomenon better than the baselines and it reaches a higher reward.

References

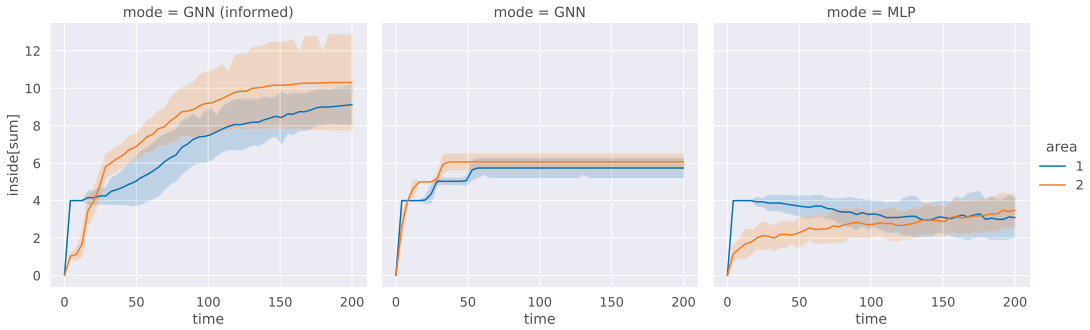
[1] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm*



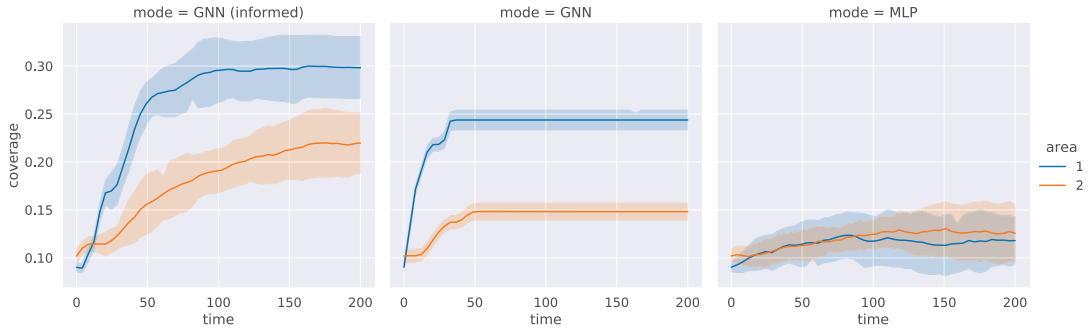
(a) Ratio of coverage of the phenomena. Our FIRL approach can outperform other approaches lacking field information.

(b) Number of agents inside the phenomenon in the three types of experiments

Fig. 5: Quantitative test results. The proposed approach can cover and track the phenomenon better than the baselines.



(a) *Two Zones* experiment: aggregated number of agent inside each phenomenon



(b) *Two Zones* experiment: ratio of covered to the uncovered zones both phenomena

Fig. 6: Coverage of two zones using the different modes of the controller.

Intell., vol. 7, no. 1, pp. 1–41, 2013.

[2] D. Bajovic, A. Bakhtiarnia, G. Bravos, and et al., "Marvel: Multimodal extreme scale data analytics for smart cities environments," in *Conf. on Communications and Networking*. IEEE, 2021, pp. 143–147.

[3] D. Pianini, F. Pettinari, R. Casadei, and L. Esterle, "A collective adaptive approach to decentralised k-coverage in multi-robot systems," *ACM Trans. on Auton. and Adapt. Systems*, vol. 17, no. 1-2, pp. 1–39, 2022.

[4] Y. Zhou, F. R. Yu, J. Chen, and Y. Kuo, "Cyber-physical-social systems: A state-of-the-art survey, challenges and opportunities," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 389–425, 2019.

[5] R. Casadei, "Macroprogramming: Concepts, state of the art, and opportunities of macroscopic behaviour modelling," *CoRR*, vol. abs/2201.03473, 2022.

[6] G. Aguzzi, "Research directions for aggregate computing with machine learning," in *Proc. of the Int. Conf. on Autonomic Computing and Self-Organizing Systems*. IEEE, 2021, pp. 310–312.

[7] G. Aguzzi, R. Casadei, and M. Viroli, "Machine learning for aggregate computing: a research roadmap," in *Proc. of the Int. Conf. on Distributed Computing Systems*. IEEE, 2022, pp. 119–124.

[8] ——, "Towards reinforcement learning-based aggregate computing," in *Proc. of the Int. Conf. on Coordination Models and Languages*. Springer, 2022, pp. 72–91.

[9] J. Beal, D. Pianini, and M. Viroli, "Aggregate programming for the internet of things," *Computer*, vol. 48, no. 9, pp. 22–30, 2015.

[10] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, and et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, 2015.

[12] C. Castelfranchi, G. Pezzulo, and L. Tummolini, "Behavioral implicit communication (BIC): communicating with smart environments," *Int. J. Ambient Comput. Intell.*, vol. 2, no. 1, pp. 1–12, 2010.

[13] L. Tummolini, C. Castelfranchi, A. Ricci, M. Viroli, and A. Omicini, ""exhibitionists" and "voyeurs" do it better: A shared environment for flexible coordination with tacit messages," in *Proc. of Int. Workshop on Environments for Multi-Agent Systems*. Springer, 2004, pp. 215–231.

[14] H. V. D. Parunak, ""go to the ant": Engineering principles from natural multi-agent systems," *Ann. Oper. Res.*, vol. 75, pp. 69–101, 1997.

[15] M. Schranz, M. Umlauft, M. Sende, and W. Elmenreich, "Swarm robotic behaviors and current applications," *Frontiers Robotics AI*, vol. 7, 2020.

[16] Y. Yang, "Many-agent reinforcement learning," Ph.D. dissertation, UCL (University College London), 2021.

[17] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang, "Mean field multi-agent reinforcement learning," pp. 5571–5580, 2018.

[18] L. Zheng, J. Yang, H. Cai, M. Zhou, W. Zhang, J. Wang, and Y. Yu, "Magent: A many-agent reinforcement learning platform for artificial collective intelligence," in *Proc. of the Conf. on Artificial Intelligence*. AAAI Press, 2018, pp. 8222–8223.

[19] C. W. Warren, "Global path planning using artificial potential fields," in *IEEE Conf. on Robotics and Automation*, 1989.

[20] M. Mamei, F. Zambonelli, and L. Leonardi, "Co-fields: A physically inspired approach to motion coordination," *IEEE Pervasive Computing*, vol. 3, no. 2, pp. 52–61, 2004.

[21] M. Viroli, J. Beal, F. Damiani, and D. Pianini, "Efficient engineering of complex self-organising systems by self-stabilising fields," in *2015 IEEE SASO conference*, 2015, pp. 81–90.

[22] G. Audrito, M. Viroli, F. Damiani, D. Pianini, and J. Beal, "A higher-order calculus of computational fields," *ACM Trans. Comput. Log.*, vol. 20, no. 1, pp. 5:1–5:55, 2019.

[23] M. Viroli, J. Beal, F. Damiani, G. Audrito, R. Casadei, and D. Pianini, "From distributed coordination to field calculus and aggregate computing," *J. Log. Algebraic Methods Program.*, vol. 109, 2019.

[24] T. D. Wolf and T. Holvoet, "Designing self-organising emergent systems based on information flows and feedback-loops." IEEE Computer Society, 2007, pp. 295–298.

[25] R. Casadei, M. Viroli, G. Aguzzi, and D. Pianini, "Scafi: A scala dsl and toolkit for aggregate programming," *SoftwareX*, vol. 20, p. 101248, 2022.

[26] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. of the 34th International Conference on Machine Learning*, 2017, pp. 1263–1272.

[27] G. Pellegrini, A. Tibo, P. Frasconi, A. Passerini, and M. Jaeger, "Learning aggregation functions," *arXiv preprint arXiv:2012.08482*, 2020.

[28] D. Silver, A. Huang, C. J. Maddison, and et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[29] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2015.

[30] C. Leiter, R. Zhang, Y. Chen, J. Belouadi, D. Larionov, V. Fresen, and S. Eger, "Chatgpt: A meta-analysis after 2.5 months," *CoRR*, vol. abs/2302.13795, 2023.

[31] K. Zhang, Z. Yang, and T. Basar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *CoRR*, vol. abs/1911.10635, 2019.

[32] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. of the Annual Conf. on Neural Information Processing Systems*, 2017, pp. 6379–6390.

[33] Y. Wu, Y. Li, Z. Wang, Y. Zhang, and T. Zhang, "More centralized training, still decentralized execution: Multi-agent conditional policy factorization," *arXiv preprint arXiv:2209.12681*, 2022.

[34] Y. Song, Y. Li, Z. Wang, Y. Zhang, and T. Zhang, "Ctds: Centralized teacher with decentralized student for multi-agent reinforcement learning," *arXiv preprint arXiv:2203.08412*, 2022.

[35] ——, "Centralized training with hybrid execution in multi-agent reinforcement learning," *arXiv preprint arXiv:2210.06274*, 2022.

[36] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang, "Mean field multi-agent reinforcement learning," in *Proc. of the Int. Conf. on Machine Learning*. PMLR, 2018, pp. 5571–5580.

[37] N. Frikha, M. Germain, M. Laurière, H. Pham, and X. Song, "Actor-critic learning for mean-field control in continuous time," *arXiv preprint arXiv:2303.06993*, 2023.

[38] A. Sosic, W. R. KhudaBukhsh, A. M. Zoubir, and H. Koeppl, "Inverse reinforcement learning in swarm systems," in *Proc. of AAMAS*. ACM, 2017, pp. 1413–1421.

[39] D. S. Bernstein, S. Zilberstein, and N. Immerman, "The complexity of decentralized control of markov decision processes," in *Proc. of the Conf. in Uncertainty in Artificial Intelligence*, 2000, pp. 32–37.

[40] E. I. Tolstaya, F. Gama, J. Paulos, G. J. Pappas, V. Kumar, and A. Ribeiro, "Learning decentralized controllers for robot swarms with graph neural networks," in *Proc. of the Conf. on Robot Learning*, 2019, pp. 671–682.

[41] E. Tolstaya, F. Gama, J. Paulos, G. Pappas, V. Kumar, and A. Ribeiro, "Learning decentralized controllers for robot swarms with graph neural networks," in *Proc. of the Conf. on Robot Learning*. PMLR, 2020, pp. 671–682.

[42] W. Gosrich, S. Mayya, R. Li, J. Paulos, M. Yim, A. Ribeiro, and V. Kumar, "Coverage control in multi-robot systems via graph neural networks," in *Proc. of the Int. Conf. on Robotics and Automation*. IEEE, 2022, pp. 8787–8793.

[43] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *CoRR*, vol. abs/1903.02428, 2019.

[44] M. Wang, D. Zheng, Z. Ye, and et al., "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv preprint arXiv:1909.01315*, 2019.

[45] J. Zhou, G. Cui, S. Hu, and et al., "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[46] B. Knyazev, G. W. Taylor, and M. R. Amer, "Understanding attention and generalization in graph neural networks," in *Conf. on Advances in Neural Information Processing Systems*, 2019, pp. 4204–4214.

[47] D. Pianini, S. Montagna, and M. Viroli, "Chemical-oriented simulation of computational systems with ALCHEMIST," *J. of Simulation*, vol. 7, no. 3, pp. 202–215, 2013.

[48] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[49] A. Paszke, S. Gross, F. Massa, and et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," https://pytorch.org, 2019.

[50] D. Kim and A. Oh, "How to find your friendly neighborhood: Graph attention design with self-supervision," *CoRR*, vol. abs/2204.04879, 2022.