Learning Opportunities in Collective Adaptive Systems

Chapter 10
Learning Opportunities in Collective Adaptive Systems

Gianluca Aguzzi⋆, Roberto Casadei⋆, Stefano Mariani†, Mirko Viroli⋆, and Franco Zambonelli†

Abstract In collective systems a multitude of computational agents coordinate to achieve a system goal beyond their individual capabilities. These systems are typically deployed in dynamic and partially unknown environments, where system designers cannot anticipate all potential situations, events, and faults that agents may experience. For this reason such systems are often *adaptive*, that is, able to change their behaviour to tolerate contingencies or embrace novel opportunities—becoming *Collective Adaptive Systems* (CAS). When engineering CAS it is crucial for the designer to take into account various essential aspects, such as deployment strategies, coordination policies for distributed execution, and the application logic itself. For each of these, *learning* could be a precious tool at designers' disposal, as it enables both design-time support and run-time adaptation with minimal a-priori knowledge. Therefore, in this chapter, we first provide a brief overview of how learning has been applied in CAS so far. Then, we describe a few novel opportunities. Finally, we discuss potential future applications of learning, particularly within the context of the Fluidware vision for pervasive systems programming.

10.1 Introduction

A Fluidware system is a specific example of a Collective Adaptive System (CAS). Adapting the definition from [8], CAS are distributed systems featuring multiple computational entities (e.g. agents) that do not have global knowledge about the system state. To achieve collective goals, these entities must interact with each other to either avoid interference or help each other, as they need information beyond their local observation capabilities.

⋆ University of Bologna, Cesena, Italy
† University of Modena and Reggio Emilia, Reggio Emilia, Italy

io metterei già nell'intro il quadro di riferimento suggerito da Roby Within a Fluidware system, there exist three key constituents: the *funnel process* abstraction, the Fluidware middleware, and a computing infrastructure (e.g., a cloud-edge system). At the core of this framework resides the notion of the funnel process—a versatile abstraction crucial for crafting collaborative applications. This funnel process operates as both a procedural entity and an information flow consumer, abstracting the complexities associated with individual devices. Its capabilities encompass capturing, merging, splitting, and replicating these information flows, thereby facilitating seamless manipulations across spatial and temporal dimensions throughout the entire infrastructure. Concurrently, the middleware works as the carrier of the complete collective application logic. Specifically, it undertakes the orchestration of funnel placements within the edge-cloud continuum, the scheduling the computations at opportune nodes, and the coordination of data movement among diverse funnels.

Adaptation within the Fluidware system manifests at two pivotal levels: at the application level of funnel processes and at the middleware level. At the application level, adaptation pertains to the dynamic adjustment of individual funnel processes to align with changing environmental conditions. For instance, consider a scenario where a Fluidware system is deployed in a smart building. As occupancy patterns shift throughout the day, the funnel processes responsible for collecting and processing sensor data from various rooms adapt their parameters to optimize energy consumption, maintaining a comfortable environment while minimizing wastage. On the other hand, adaptation at the middleware level involves the reconfiguration of the Fluidware middleware to accommodate shifts in system requirements or resource availability. Take the case of a Fluidware system used for traffic management in a smart city context. During periods of heavy traffic congestion, the middleware might dynamically allocate funnel processes to elaborate and analyse incoming data streams from traffic sensors in real-time. As traffic conditions normalize, the middleware could scale down these resources to avoid unnecessary overhead. However, achieving *effective* adaptation in such systems is complex due to the highly dynamic and unpredictable environment in which each agent operates. Centralized or hierarchical adaptation is often not feasible, as it depends on unrealistic assumptions such as the ability of a single controller to obtain a consistent and complete system-wide view and to process this information, reason, and distribute decisions in time. Instead, recent research has demonstrated that *decentralized self-adaptation* or *self-organization* can overcome issues associated to centralized adaptation. This paradigm shift increases resilience, improves scalability, and eliminates single points of failure.

To address these challenges, agents could be enhanced with *learning* capabilities, allowing them to instantiate and refine learning models using the knowledge acquired from observing their environment and peers. These learning models provide the means to improve quality attributes such as performance and cost for an individual agent or the entire collective. However, learning in CAS is challenging since there are several levels that need to be considered, starting from how to *deploy* a concrete collective application, how the middleware should execute it *efficiently*, there including communication and coordination overhead, and how the application

can be engineered. Furthermore, CAS systems are obviously engineered by software designers and developers, hence learning could also be used to assist them at design-time.

Accordingly, in this chapter, we first briefly overview the state of the art in CAS learning (Section 10.2), then we describe a few novel approaches developed within the Fluidware project to push forward its vision (Sections 10.4,10.3,10.6,10.5), and finally we discuss further opportunities to plug learning into the Fluidware vision (Section 10.7).

10.2 State of art

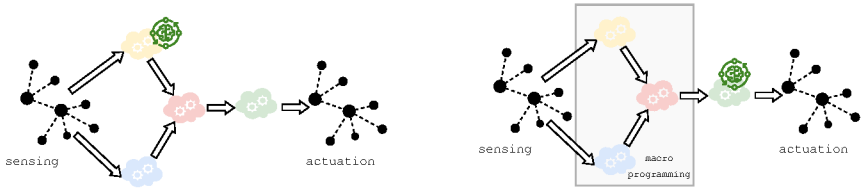
In recent years, there has been significant development in the field of machine learning techniques, both in supervised (e.g., AlexNet and similar) and unsupervised (LLM) mode, as well as reinforcement learning (RL) and their combinations (e.g., ChatGPT). Among the many techniques, Multi-agent reinforcement learning (MARL) it is one of most promising: it is a sub-field of RL that focuses on studying the behaviour of multiple learning agents that coexist in a shared environment. Each agent is motivated by its own rewards, and does actions to advance its own interests; in some environments these interests are aligned with the interests of other agents, resulting in *cooperative* or *collective* behaviour. MARL is a crucial research area that aims to develop robust, scalable, and resilient multi-agent systems. Numerous approaches, algorithms, and solutions have been proposed in the literature over the years to address various challenges and opportunities in MARL. However, MARL has a wide range of applications in different domains, which motivates different perspectives and classifications of MARL algorithms. Therefore, several surveys have been conducted to provide comprehensive overviews of specific aspects or categories of MARL algorithms. In the following, we present the relevant surveys in the context of Fluidware applications:

1. *On Learning in Collective Self-Adaptive Systems: State of Practice and a 3D Framework [8]* – In this survey, the authors focus on learning in the context of collective adaptive systems. They identify the main algorithms used (reinforcement learning, supervised learning, unsupervised learning) and create a framework to classify different applications with respect to autonomy (full vs. restricted), behavior (selfish vs. altruistic), and knowledge access (maximal vs. minimal).
2. *A Survey of Self-Organization Mechanisms in Multiagent Systems [47]* – This survey explores how self-organization is accomplished and used in CAS. The authors highlight how self-organization can assist Reinforcement Learning to converge to good behaviors, reducing the problem of concurrent learning in MAS/CAS.
3. *Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms [50]* – This recent survey delves into MARL settings for both competitive and cooperative tasks, focusing on the theoretical background. It also

discusses network and many-agent settings, with an emphasis on mean-field solutions and their convergence guarantees.

4. *Decentralized Multi-Agent Reinforcement Learning with Networked Agents – Recent Advances [51]* – This survey primarily focuses on network MARL studies in both cooperative and competitive settings. It does not cover CTDE (centralized training and decentralized execution) as learning in such scenarios should be performed online without a central authority to aggregate the system view and improve shared policies. The scale is not the primary concern here, but rather the decentralization. In Fludiware, scale becomes important as we generally cannot know the whole population of the system.
5. *A Survey and Critique of Multiagent Deep Reinforcement Learning [18]* – In this paper, the authors concentrate on deep learning aspects, highlighting the advantages of MARL in various settings, including cooperative, competitive, and mixed scenarios. They categorize the main classes as Analysis of Emergent Behavior, Learn to Communicate, and Agents Modeling Agents. While they don't primarily focus on many-agent aspects, they do discuss them in the context of few agents.
6. *A Review of Cooperative Multi-Agent Deep Reinforcement Learning [29]* – This paper solely focuses on the cooperative aspect of deep MARL. The algorithms are classified based on how agents learn, leading to independent learners, full observable critics, value decomposition functions, and consensus methods. While the main goal is not large-scale scenarios, they also discuss some work related to many-agent approaches like mean field.
7. *A Survey of Multi-Agent Reinforcement Learning with Communication [52]* – This survey mainly centres on communication in the context of MARL. It discusses the design perspective, including the learning scheme, agent communication, channel construction between agents, and policy development, among other approaches discussed in the literature.
8. *Multi-Agent Deep Reinforcement Learning: A Survey [15]* – Another survey focusing on the MARL perspective but with a different classification. It first discusses the learning schemes (Centralized, Decentralized, Centralized Training, and Centralized Execution) and addresses the challenges of the current landscape as of 2019.
9. *Cooperative Multi-Agent Learning: The State of the Art [30]* – An older reference of MARL in the context of cooperative tasks. The survey explores different types of policies (homogeneous vs. heterogeneous) and various ways of learning (team-based vs. concurrent) in cooperative multi-agent settings.
10. *Multiagent systems: A survey from a machine learning perspective [42]* - Another older but seminal reference, actually one of the first to consider learning in MAS. It categorises how learning affects MAS under different criteria (homogeneous vs. heterogeneous agents, communication Vs. no-communication).

Focusing solely on CAS systems, learning can be applied at different major levels: i) at the *application* level (i.e., inside the funnel processes), ii) at the *middleware* level, and iii) at the *deployment* level [2]. Also, it can be used at design or run time, for either supporting CAS developers and administrators, or directly inject adaptiveness



(a) Funnel synthesis (b) Program learning

Fig. 10.1: High level overview of the two approaches.

in the distributed components. At the application level, learning is used directly to solve collective tasks. For example, in the context of swarm robotics, it involves the ability of a group of agents to form a flock [41]. At the middleware level, the focus is on solving non-functional aspects. This includes reducing distributed computing consumption or minimizing the number of exchanged messages. At the deployment level, research focuses on how to opportunistically move computation in complex network infrastructures, such as cloud-edge-fog environments.

Reinforcement learning is extensively utilized at all levels in these systems due to its flexibility, online adaptability, and scalability. The focus of this chapter is mainly on reinforcement learning, and in particular multi-agent reinforcement learning, as it naturally embraces a *distributed collective* of learning agents, while considering their strict relationship with the environment—as collective adaptive systems are. Closely related to reinforcement learning is another form of learning that recently got increasing attention from the RL and MARL communities: *causal learning* [17]. Causal learning is a discipline that studies how to learn causal models of an environment or phenomenon of interest through *interventions* [37], that is, through interaction with such an environment (or, phenomenon)—as (MA)RL. The main difference with respect to RL – as well as other forms of statistical machine learning, such as supervised and unsupervised learning – is that causal learning is able to tell apart cause-effect relationships between variables from mere associations: traditional ML cannot give this guarantee by construction (see work from Judea Pearl and other, such as [39]). Section 10.5 gives an example.

10.3 Learning along funnel processes

Paper on collective program synthesis (i.e., [3]) In Fluidware systems, the application logic is defined as a collective program, which is then executed by the middleware. The way in which this program is defined is crucial for the overall performance of the system. Among the many approaches, macro programming is one of the most promising: The focus is switched from the individual agent to the collective, having a top-down global-to-local approach and having the ability to combine simple behaviors to create complex ones – as exemplified by the aggregate computing

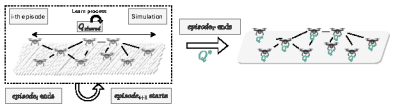
paradigm [44]. However, in practice, constructing these aggregate computing applications is anything but straightforward. This is because, even if the paradigm support adaptive computation, the programmers must have a deep understanding of the environment in which the system will be deployed. Indeed certain basic components/behaviour work well in specific environmental dynamics but not in others, thus leaving the burden on the programmer to choose the most suitable combinations of blocks for a given environment. To address this issue, modern approaches focus on combining parts of macro-programming paradigm with machine learning techniques. The idea is to preserve the characteristics of macro programming, such as declarativeness and composability, while incorporating those of machine learning, including adaptability and the ability to solve complex programs through "by-doing" or "by-example" methods.

There are various ways in which this combination can be achieved. For instance, in [3], "pieces" of a partially specified program can be filled in, resembling the field of program synthesis. On the other hand, in [4], the result of the macro program is used to guide the learning process, enabling more agile synthesis of the collective controller (Figure 10.2 gives an overview of both ideas).

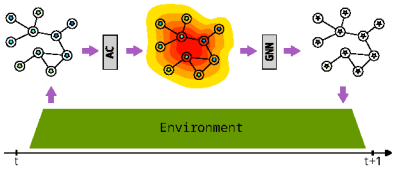
Focusing on the foster approach, giving a complex network of funnels devising with macro programming, the idea is to learn only the parts of the whole network that are: i) hard to define, since they are inherently complex, or ii) strictly dependent on the environment in which the system is deployed. Therefore, the programmer will leave some funnel processes undefined, and the learning algorithm will fill in the gaps, in a process that is similar to program *synthesis*/*sketching*. In [3], the authors propose a variant of Q-learning that can be used to synthesize collective distributed programs basing part of the program on aggregate computing paradigm. Particularly, the learning process is performed at simulation time, where the agent collect *local* experiences in a *global* reply buffer. During the learning process, the agent executes the program as usual, but when it reaches a "sketching" point, it will execute a random action with a certain probability, otherwise it will execute the action suggested by the Q-table. The Q function is shared and updated globally, but agents can only observe their local state and the messages received from their neighbors. That means that, once the Q function is learned, each agent can execute the program locally, without the need of a central authority. The buffer is then used to refine a Q-table with the usual Q-learning algorithm. This is training procedure is known as centralised training and decentralised execution (CTDE) [13] (Figure 10.2a). The approach is then used to successfully learn a correction policy in order to speed up the convergence of a self-stabilizing algorithm.

In [4], the authors propose a different approach, called *Field-informed Reinforcement Learning* (FiRL), where the program computed by the macro programming paradigm is used to guide the learning process. Particularly, the program is used to enrich the state space of the learning algorithm with collective feature, ease up the learning process, and improve the quality of the learned policy (Figure 10.2b). Indeed, in MARL algorithm the communication is often implicit, and the agents must learn how to communicate with each other to achieve the desired goal. Adding that information to the state space, the agents do not need to learn how to communicate,

but only to leverage the information provided by the program. FiRL is based on Deep Q-learning, where the Q function is approximated by a *graph neural network* (GNN). The choice of the GNN was motivated by the fact that it generalize well to unseen topologies, and can be used locally, even if it can be trained from the whole graph during the training phase. This combination of macro programming and learning is used to learn a collective controller for a swarm of robots, where the goal is to cover a moving phenomenon (e.g., a wildfire) that is not known a priori and spans a large area, meaning that no single agent can cover it alone and can detect the whole phenomenon.



(a) Overview of the training procedure.



(b) Overview of Field-informed Reinforcement Learning

Fig. 10.2: Overview of the two approaches.

10.4 Learning scheduling policies to optimize middleware

Paper on scheduling in CAS (i.e., [1]) A crucial aspect for the Fluidware middleware is the scheduling of the funnel processes, this is particularly import in macro programming, as it determines how the collective program is executed. Some of these macro approaches, such as Buzz [33] and aggregate computing [44], employ a *proactive* execution model that breaks down distributed execution into rounds typically composed of three macro phases:

1. *creation of an evaluation context*: in this phase, the necessary information is gathered to evaluate the collective program (e.g., messages from neighbors, sensory information, previous state).
2. *program evaluation*: given the context, each node evaluates the defined collective program, creating the desired output.
3. *actuation and message sharing*: with the produced output, nodes can perform actions that modify the environment. Furthermore, the necessary messages are exchanged among neighbors to evaluate the defined program.

The various models do not specify "when" these phases should occur, thereby leaving some decision-making margin to the middleware. In fact, tuning the evaluation frequency can impact various aspects, including energy consumption and convergence times to the defined collective structure.

One of the earliest works focused on programming distributed schedulers [5], where rules expressed with aggregate programming allowed the description of schedulers that react to environmental/temporal events. However, despite the interest in this approach, there was the problem of having to program these behaviors in advance, with the disadvantage of potentially overlooking unrecognized environmental dynamics and making the controller definition hard.

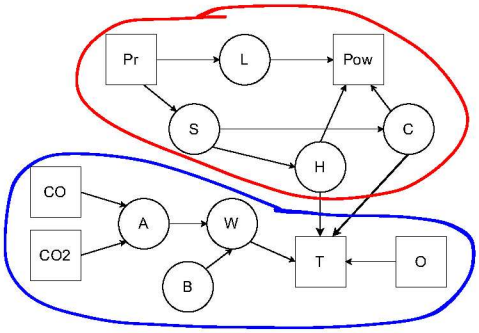
Therefore, in [1], an attempt was made to create these distributed schedulers through reinforcement learning – particularly a variant of Q-Learning – with the goal of optimizing a certain family of distributed computations, namely *self-stabilizing* computations – that are programs which have the characteristic of achieving a stable output given a stable context. In the proposed approach, agents are equipped with a Q-table that is used to decide when to evaluate the collective program, i.e., the action space consist of the next wake-up time. As in the previous work, the agents collect local experiences in a global reply buffer, and the Q-table is updated using the CTDE paradigm. The simulations show that the learning algorithm is able to reduce the energy consumption of the system, while maintaining the same convergence time of the self-stabilizing algorithm.

10.5 Learning the environment dynamics

In scenarios where multiple sensor and actuator devices have many inter-dependencies, it is impractical (if not impossible) for the system designer to foresee (i) all the possible relations between the environmental variables, and (ii) all the possible effects of acting on them. Hence, in a Fluidware perspective, it would be very difficult to design funnel processes with guarantees to produce desired results (e.g. in terms of affecting the environment in a desired way). Yet, for funnel processes and software agents alike, to have explicit knowledge of the network of cause-effects relations (aka of the causal model of the agents-environment system [31]) is of fundamental importance to make sense of their operational environment, autonomously decide plans of actions, and being able to explain such decisions [6, 25]. Thus, having software agents (such as funnel processes) autonomously *learn* such causal model once deployed in the target (simulated) environment, with minimal apriori knowledge, may be extremely valuable.

Causal learning techniques [16] enable distinguishing causal relations from mere associations. For instance, recognizing that the states of the air conditioning system and of the temperature in a room are not simply correlated, but that the first causes the second to change. The result of such causal learning process is thus a *causal network* relating causes and effects with directed edges, a task that has been shown to be out of reach for statistical machine learning models [39], which rely on purely *observational* data to learn *associations*. It is worth noting here that also funnel process define a network of processing steps, in a sense, hence can be expressed in terms of causal graphs: some nodes will be variables, some will be operators

(a) Two agents with limited observability.



(b) The ground truth causal network.

Fig. 10.3: Partial observability in a CAS.

manipulating such variables. Learning a causal model would then correspond to automatically assembling funnel processes.

In most existing proposals, though, causal learning assumes a *centralized* setting, with a single learner/agent having full observability and control over the environment variables, or with a central site aggregating distributed observations or merging local models [16, 45, 17]. However, the centralized assumption can rarely hold for many real-world deployments, and especially for CAS, where multiple agents are typically deployed in different regions of an environment. There, each agent only has partial observability of the environmental variables, and the capability of affecting only a limited portion of the overall environmental variables through its own actuators (Figure 10.3). Hence, authors of [28] propose a *multi-agent protocol for distributed learning of causal networks*, aimed at letting agents in a MAS (but could be a CAS, or a Fluidware system with funnels instead of agents) cooperate in unveiling causal relationships that individuals could not reveal by themselves, due to partial observability.

They consider N agents $\mathcal{A} = \{\mathcal{A}_i\},\ i = 1, \ldots, N$ willing to learn a causal network M_i of the relationships between V environment variables. The set V is partitioned into two (possibly, empty) sets, one of *controlled* variables C (i.e. corresponding to actuators) and one of *uncontrolled* variables U (i.e. sensors), such that $V = C \cup U$. It is assumed that each agent knows an algorithm for *independently* learning its own local causal network $\mathcal{L}_i$, that is, the causal network learnt by relying solely on its own known variables P_i (i.e. without running our multi-agent protocol). It doesn't really matter which one, but they choose one already adopted in an agent-oriented scenario [12]. Each agent has *partial observability of the environment*, that is, is aware of only P_i out of the V variables. Formally: $P_i = C_i \cup U_i,\ P_i \subset V \Rightarrow C_i \subset C, U_i \subset U$. However, no variable in V can be unknown to every agent, that is: $V = \bigcup_{i=1}^{N} P_i$—note that an overlap between the different P_i is admitted. M_i is not necessarily the *global* causal network G, that is, the one including every variable in V and every link

amongst them. Rather, it is the "minimal" causal network that each agent "needs" to correctly understand situations and plan actions. That is, M_i contains all the variables in P_i and their links, plus the subset of variables in $V \setminus P_i$ that have links with variables in P_i.

The proposed protocol revolves around agents exchanging observational data and carry out *interventions* whenever possible, that is, changing the values of their controlled variables through the simulation software that provides access to virtual devices (iCasa is used [22]). For instance, an agent may repeatedly (at a given time interval, during a given time period) activate and de-activate the A/C to check (or let other agents check) whether the temperature changes as a consequence. Interventions are the fundamental mechanisms to check whether two variables are linked in a cause-effect relationship and not by a mere correlation [32]. Intuitively, an intervention deliberately changes the value of a variable, *all others being untouched*, to see whether it affects others. As such, interventions require the ability to *control* variables, as is the case of the variables expressing the status of actuators in agent systems.

Authors show that multi-agent causal learning has superior accuracy with respect to the single-agent case, as it can learn more causal relationships while doing less mistakes (causal links learnt but not present in the ground truth)—Figure 10.4.

In Fluidware, causal graphs such as the one shown in Figure 10.4 may guide funnel processes' own computations. On the one hand, they can use the graph to decide where to pick data from to make a particular decision (e.g. check CO levels to decide what to do with the window W). On the other hand, the same graph can also be used to predict outcomes of actions ("what if we close the window?"). In summary, causal graph are a flexible and powerful tool to make decisions, and learning them is a great enabler of adaptation (if the environment changes a new graph can be learnt to cope with it).

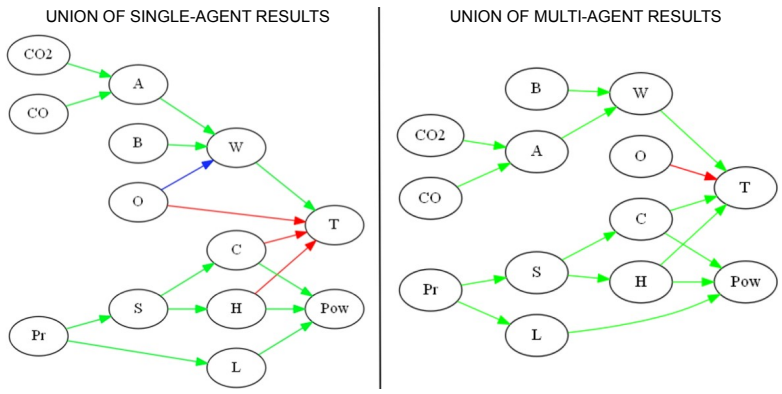


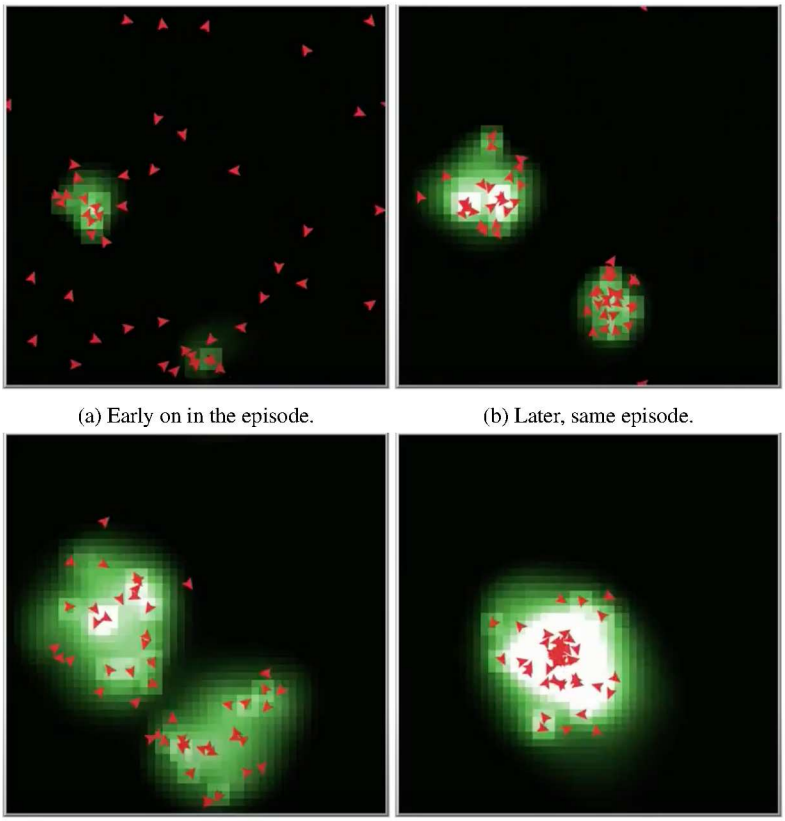
Fig. 10.4: Multi-agent learning outperforms single-agent causal discovery: it makes less mistakes (red arrows) and finds less false causal relations (blue arrows)—green arrows are correct.

10.6 Learning local communication for collective goal achievement

In CAS where agents' observability of each other actions and their effects has no limits, coordination may be implicit [23]: each agent reacts to what others are doing as best as it can. In more challenging scenarios, however, *explicit* communicative actions are needed: agents need to tell each other what to do. Such communication actions are often assumed to be among the built-in capabilities of agents, and the communication policies resulting from their composition are defined according to the specific goal to achieve. However, *learning* such policies would enable agents to adapt to new deployments, run-time changes, and/or changing goals with minimal designers' intervention [34, 24]. The same applies to Fluidware, at multiple levels: at the application level, different funnel processes with different functions can be composed (akin to service composition) to achieve the system goals, and such a composition may require communication between them, that could be learnt; at the middleware level, handling distributed event flows to feed funnel processes as well as their composition requires processing nodes to communicate, and, again the employed protocol could be learnt.

Multi-Agent Reinforcement Learning (MARL) provides methods for learning to communicate, either as a form of language (ungrounded communication) or with practical environmental actions (grounded communications). Stigmergic communication via *pheromone* is a kind of grounded communication: a biological marker is deposited to persist for some time and diffuse within some range (thus can be sensed by others locally in time and space). Such a mechanism is pervasively used in natural systems [19], in MAS, and swarm robotics as well [27, 20, 49] for successful self-organised behaviours to emerge. Ant foraging is an example, with the Ant Colony Optimisation algorithm for distributed optimisation [11], while termites' nest building [10] and slime mold aggregation [38], that we use as our scenario below, are others. Slime mold cells [36] (the agents) have innate policies ("rules") to deposit pheromone (i.e. *when*, in what circumstances), and to follow it (e.g. instead of wandering randomly) with the individual goal to stay close to as many other slime mold cells as possible, contributing to the global one of assembling clusters—through self-organisation.

The literature reports an attempt at learning such policies from scratch, that is, *when* to deposit and follow pheromone, whereas most of related literature in related fields, e.g. evolutionary computation, deals instead with *how much* pheromone to deposit. Authors of [**?**], in fact, adopt independent Q-learning in a slime mold cells MARL environment where agents have partial observability (e.g. they perceive communications within a very limited range) and achieve a number of results (see Figure 10.5 as reference). First, pheromone-based communication is learnt consistently despite non-stationarity; second, the learnt policies are quicker to converge to clusters than the built-in rules usually adopted in artificial swarms; third, agents do learn to self-organise to best achieve the ascribed systemic goal, by coordinating action selection so that pheromone deposit happens in a few places (to have a



(a) Early on in the episode. (b) Later, same episode.

(c) Still same episode, later on. (d) End of episode.

Fig. 10.5: Agents learn a collectively effective distribution of communicative actions: (a) few spam pheromone (to form a few clusters), (b,d) most follow pheromone trails (to aggregate), (c) few persist to roam randomly (to take the chance of merging clusters).

few bigger clusters) and persists in time (to not disperse them); finally, learning to communicate is also able to adapt to different collective goals, as agents do adapt by learning new policies with minimal external intervention (e.g. by solely changing the reward function).

Figure 10.5 shows an example run: in later stages of training, agents (red triangles) have learnt how to exploit pheromone-based communication to achieve the clustering goal. In fact, agents have learnt how to collectively perform the correct actions by distributing with the right balance the actions to make, in three fundamental ways: (1) few deposit pheromone steadily while not moving at all, to communicate to others their presence; (2) most of the agents look around searching for pheromone, that is, are eager to listen to "messages" (pheromone deposit) coming from others,

so as to promptly move in their direction; (3) another few still roam randomly, ignoring clustering messages (pheromone trails) to check whether better clustering opportunities exist. Without (1), for instance if too many agents spam pheromone, it may happen that agents try to cluster in too many different locations. Without (2), that is if too few agents react to pheromone trails, clusters will not reinforce enough and may never form. Without (3), it would be impossible for behaviours like the one depicted in Figure 10.5 to happen: merging clusters to form a bigger one would be very difficult as agents would always prefer gravitating around existing ones.

In the context of Fluidware, these results demonstrate that it is possible and effective to learn communication policies (or, composition alike) even in CAS with many agents, hence despite non stationarity. By learning communication policies at run-time, with MARL, it would be possible to adapt to contingencies both at the middleware level (e.g. broken communication links or delayed nodes) and at the application level (e.g. funnel processes reconfiguring their composition for a new goal to pursue).

10.7 Relationship with Fluidware

qui ci sta bene anche una figura che fa vedere i vari punti in cui può avvenire l'apprendimento

The key innovative idea of Fluidware is to abstract collectives of devices of the IoT fabric as *sources, digesters, and targets* of *distributed "flows" of contextualized events*, carrying information about data produced and actuating commands. Accordingly, programming services and applications implies *declaratively specifying "funnel processes" to channel, elaborate, and re-direct such flows* in a fully-distributed way, as a means to coordinate the activities of devices and realize services and applications.

This overview already gives clues about where learning could be applied within Fluidware based on its cornerstone abstractions and mechanisms. First of all, Fluidware decouples devices from event/data streams, hence learning what device should be feeding or consuming which stream is an opportunity to consider. Second, such streams must be connected to devices and processing nodes acting as prosumers of events: how to connect them (in what topology, with what communication protocol, etc.) is another potential target for learning. Finally, an perhaps most naturally, learning can be utilized to synthesize, refine, or adapt the "behaviour" of funnel process, being them the active processing component in Fluidware paradigm devoted to purse goals and possibly perform autonomous decision making.

Besides these, other opportunities for learning may be devised out by considering the application scenarios that Fluidware aims to tackle, and their typical challenges and requirements. The potential high number and density of *deployed devices* raises issues about communication topology and fault tolerance: what if devices move? what if network links break? Finding the best way to communicate may be a task apt for learning. The need to operate on top of a dynamically *evolving infrastructure*

comprising IoT/edge/cloud devices and resources raises deployment issues: where to place running process? how to best relocate them at run-time to cope with processing load fluctuations or hardware failures? Such a deployment policies may be another likely target for learning. Finally, the need for goal-oriented *orchestration* of distributed activities across devices of heterogeneous computational power (from tiny devices to cloud servers) raises issues about coordination and cooperation of distributed autonomous components for which MARL is particularly suited for.

The following sections elaborate on these opportunities.

10.7.1 Funnel processes that learn

Funnel processes are, essentially, the "algorithms" that the application designer defines and the Fluidware platform runs to achieve the application goals. Then, a first learning opportunity comes from *program synthesis* with the *sketching* technique [40] also suggested in [2] and experimented in [1]: ask the developer to only specify a program "blueprint", e.g. a loose specification of the "high-level" expected behaviour with missing lower-level details, that will be completed through learning—e.g. action policies learnt by reinforcement learning, or data acquisition operations learnt via prediction models.

Another learning opportunity comes when considering that funnel processes may be *composed* to modularly build more complex functionalities on top of simpler ones. Such compositions may explicitly designed or learnt [35], an increasingly adopted approach to deal with the complexity and uncertainty on dynamic software environments. In [26], for instance, reinforcement learning is used in a multi-agent setting to learn the most appropriate matching between requests for services and services providers, also considering how to compose existing services in novel ways when a readily available service cannot meet the service request's requirements.

However, service composition does not account for any arbitrary *coordination* of funnel processes towards achievement of complex system goals. For instance, it may be the case that access to a given events flow must be given to a single funnel process at a time, hence a distributed mutual exclusion protocol must be enacted to coordinate competing funnels. Or, funnel processes may need to learn how to coordinate on the fly to perform adaptation actions in face of unexpected system disruption. Learning to coordinate, either implicitly or via explicit communication, is an open challenge that has seen increasing attention in recent years, especially in the multi-agent reinforcement learning community [43, 7, 48].

10.7.2 Learning to wire event flows to funnels

Funnel processes work off of event flows, that is, by consuming and producing unbounded streams of events stemming from the data emitted by deployed devices.

Such event flows are decoupled from the devices that produce them, from the standpoint of funnel processes (and their programmers). That is, they don't care which device contributed to which event flow, and how, they only care about having access to the right event at the right time. This means that at different points in time the same funnel process may actually be connected (behind the scenes) to different devices, since different devices may be feeding the same "logical" event flow—depending on run-time contingencies. Learning how to wire this connection dynamically is an interesting opportunity.

For instance, devices may produce measurements with different accuracies and confidence in their values over time, due to battery issues, signal degradation, or any other reasons. A funnel process in charge of a critical task may require events always with maximum accuracy and confidence, hence predicting a loss of accuracy may be crucial to dynamically re-wire devices to event flows.

Also, funnel processes are meant to pursue application goals, but the means to do so may change over time, wither due to new opportunities or faults. Hence, also the wiring of event flows to funnels may need learning adaptations to perform to cope with the ever changing environment typical of CAS.

10.7.3 Learn deployment policies

Another fundamental cornerstone of the Fluidware approach is *opportunistic deployment*, for achieving both scale-independence and robustness: the capability of deploying, migrating, replicating, and re-locating processing components across the Edge-to-Cloud spectrum. Fluidware in fact does not foresee exclusively Edge deployments, but welcomes a synergistic exploitation of all the tiers of a possible deployment: Edge to harness real-time processing on-board devices, Fog computing when battery life or computational power issues demand so, and Cloud when low latency isn't a constraint or heavy analytics must be carried out. However, in this complex, multi-tiered deployment scenario it could be difficult to define where to place each processing component at design-time. Given the intrinsically dynamic and unpredictable nature of CAS, it would be much better to let the Fluidware middleware learn the best deployment configurations based on past experience.

For instance, a Fluidware middleware may continuously monitor aspects such as battery life of devices, connectivity strength, processing time, data back-pressure, as well as application requirements and the like to try to predict the need for migrating or re-locating processing tasks. A device with a low battery level may stop processing data on-board and delegate to a Fog node, or in case the middleware predicts a rapid upsurge of incoming requests may decide to transfer processing to a Cloud node at the cost of latency.

All of this can be done without learning of course, and is in fact the preferred way of doing these things nowadays. However, this puts lots of burden on system designer that must carefully engineer static policies to migrate and re-locate components when

needed. Having a system able to autonomously learn when and how to reconfigure deployment at run-time would be incredibly valuable.

The more modular and fine-grained they are, the easier would be for applications to, for instance, learn how to compose them for achieving a desired goal.

10.8 Open challenges

10.8.1 Multi-agent Credit Assignment: How to Reward Agents for Their Contribution

A significant challenge in many agent system is determining how to fairly reward individual agents for their contributions to collective goals, especially when their actions are interdependent. This is an open problem typically referred to as *multi-agent credit assignment* in the literature. Even if several solutions have been proposed like difference rewards [46], ad-hoc reward shaping [9], and ad-hoc multi-agent learning algorithms (like COMA [14]), the problem is still open in the general case, and it is even more challenging in the case of CAS, where agents are not necessarily homogeneous and very numerous. This could involve the development of new credit assignment algorithms that take into account the complex interactions between agents in such large scale scenario.

10.8.2 Lightweight Learning: How to Learn with Limited Resources

In Collective Adaptive Systems (CAS) environments such as the Internet of Things (IoT), computational resources are often severely limited. Devices like sensors, actuators, and edge nodes may have constraints in terms of processing power, memory, and energy availability. In such contexts, lightweight learning algorithms become not just beneficial but essential for real-time adaptability and efficient operation. Traditional machine learning algorithms often require significant computational resources for training and inference. This is impractical in IoT settings where devices are constrained by limited processing capabilities and power supplies. Therefore, there is a pressing need for algorithms that can operate under these constraints while still providing accurate and timely insights. Even if several solutions have been proposed like federated learning [21] however, they still do not consider the specificities of CAS, where there is a need for a collective outcome. Therefore, this could involve the development of new lightweight learning algorithms that take into account the specificities of CAS, considering both the resource constraints and the emergence of desired behaviours.

10.8.3 Lightweight Inference: How to Infer with Limited Resources

Similar to the challenges posed by lightweight learning, the need for efficient inference mechanisms is equally critical in Collective Adaptive Systems (CAS), particularly in edge-cloud computing scenarios. In these settings, data is often generated at the edge of the network, close to where it is needed most. However, the edge devices, such as IoT sensors and mobile phones, are typically constrained in terms of computational power, memory, and energy resources. As a result, the system must be capable of making quick and accurate inferences with these limited resources.

One approach to tackle this challenge is the use of specialized inference engines that are optimized for low-resource environments. These engines can be designed to perform specific types of calculations more efficiently, thereby reducing the computational load on the device. For example, TinyML is an emerging field that focuses on deploying machine learning algorithms on tiny, low-power hardware. Algorithms in this domain are optimized for quick inference and low energy consumption, making them ideal for edge computing scenarios.

Another strategy is the use of quantization techniques that simplify the numerical precision of the model parameters. By reducing the bit-width of the model's weights and biases, the computational and memory requirements for inference can be significantly lowered. This comes at the cost of some loss in model accuracy, but in many real-world applications, this trade-off is acceptable.

Edge-cloud orchestration can also play a pivotal role in optimizing inference tasks. In scenarios where the edge device lacks the necessary computational power to perform the inference locally, the task can be offloaded to a nearby cloud server. The decision to offload can be made dynamically based on various factors such as the current load on the device, network latency, and the urgency of the inference task. Effective edge-cloud orchestration algorithms can help in making these decisions in real-time.

Model pruning is another technique that can be employed to make the models more lightweight. By removing the neurons that contribute the least to the model's performance, the complexity of the neural network can be reduced, thereby speeding up the inference time.

10.9 Conclusion

Collective Adaptive Systems (CAS) offer a promising avenue for tackling complex, large-scale problems that are beyond the capabilities of individual agents. However, the dynamic and distributed nature of these systems introduces a host of challenges that need to be addressed for their effective deployment and operation. This paper has provided an overview of the state-of-the-art in learning techniques for CAS, focusing on the Fluidware framework as a case study. We have also identified several open challenges that represent important directions for future research.

References

1. Aguzzi, G., Casadei, R., Viroli, M.: Addressing collective computations efficiency: Towards a platform-level reinforcement learning approach. In: Casadei, R., Nitto, E.D., Gerostathopoulos, I., Pianini, D., Dusparic, I., Wood, T., Nelson, P.R., Pournaras, E., Bencomo, N., Götz, S., Krupitzer, C., Raibulet, C. (eds.) IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2022, Virtual, CA, USA, September 19-23, 2022. pp. 11–20. IEEE (2022). https://doi.org/10.1109/ACSOS55765.2022.00019, https://doi.org/10.1109/ACSOS55765.2022.00019
2. Aguzzi, G., Casadei, R., Viroli, M.: Machine learning for aggregate computing: a research roadmap. In: 42nd IEEE International Conference on Distributed Computing Systems, ICDCS Workshops, Bologna, Italy, July 10, 2022. pp. 119–124. IEEE (2022). https://doi.org/10.1109/ICDCSW56584.2022.00032, https://doi.org/10.1109/ICDCSW56584.2022.00032
3. Aguzzi, G., Casadei, R., Viroli, M.: Towards reinforcement learning-based aggregate computing. In: ter Beek, M.H., Sirjani, M. (eds.) Coordination Models and Languages - 24th IFIP WG 6.1 International Conference, COORDINATION 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13271, pp. 72–91. Springer (2022). https://doi.org/10.1007/978-3-031-08143-9_5, https://doi.org/10.1007/978-3-031-08143-9_5
4. Aguzzi, G., Viroli, M., Esterle, L.: Field-informed reinforcement learning of collective tasks with graph neural networks. In: IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2023, CA, Toronto, September 19-23, 2022 (2023)
5. Casadei, R., Mariani, S., Pianini, D., Viroli, M., Zambonelli, F.: Space-fluid adaptive sampling: A field-based, self-organising approach. In: ter Beek, M.H., Sirjani, M. (eds.) Coordination Models and Languages - 24th IFIP WG 6.1 International Conference, COORDINATION 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13271, pp. 99–117. Springer (2022). https://doi.org/10.1007/978-3-031-08143-9_7, https://doi.org/10.1007/978-3-031-08143-9_7
6. Chou, Y., Moreira, C., Bruza, P., Ouyang, C., Jorge, J.A.: Counterfactuals and causability in explainable artificial intelligence: Theory, algorithms, and applications. Inf. Fusion **81** (2022)
7. Christoffersen, P.J.K., Haupt, A.A., Hadfield-Menell, D.: Get it in writing: Formal contracts mitigate social dilemmas in multi-agent RL. In: Agmon, N., An, B., Ricci, A., Yeoh, W. (eds.) Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023, London, United Kingdom, 29 May 2023 - 2 June 2023. pp. 448–456. ACM (2023). https://doi.org/10.5555/3545946.3598670, https://dl.acm.org/doi/10.5555/3545946.3598670
8. D'Angelo, M., Gerasimou, S., Ghahremani, S., Grohmann, J., Nunes, I., Pournaras, E., Tomforde, S.: On learning in collective self-adaptive systems: state of practice and a 3d framework. In: Litoiu, M., Clarke, S., Tei, K. (eds.) Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2019, Montreal, QC, Canada, May 25-31, 2019. pp. 13–24. ACM (2019). https://doi.org/10.1109/SEAMS.2019.00012, https://doi.org/10.1109/SEAMS.2019.00012
9. Devlin, S., Kudenko, D.: Theoretical considerations of potential-based reward shaping for multi-agent systems. In: Sonenberg, L., Stone, P., Tumer, K., Yolum, P. (eds.) 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), Taipei, Taiwan, May 2-6, 2011, Volume 1-3. pp. 225–232. IFAAMAS (2011), http://portal.acm.org/citation.cfm?id=2030503&CFID=69153967&CFTOKEN=38069692
10. Dhokia, V., Essink, W.P., Flynn, J.M.: A generative multi-agent design methodology for additively manufactured parts inspired by termite nest building. CIRP Annals **66**(1) (2017)
11. Dorigo, M., Stützle, T.: Ant colony optimization. MIT Press (2004)

12. Fadiga, K., Houzé, É., Diaconescu, A., Dessalles, J.: To do or not to do: finding causal relations in smart homes. In: IEEE International Conference on Autonomic Computing and Self-Organizing Systems (2021)
13. Foerster, J.N., Assael, Y.M., de Freitas, N., Whiteson, S.: Learning to communicate with deep multi-agent reinforcement learning. In: Lee, D.D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain. pp. 2137–2145 (2016), https://proceedings.neurips.cc/paper/2016/hash/c7635bfd99248a2cdef8249ef7bfbef4-Abstract.html
14. Foerster, J.N., Farquhar, G., Afouras, T., Nardelli, N., Whiteson, S.: Counterfactual multi-agent policy gradients. In: McIlraith, S.A., Weinberger, K.Q. (eds.) Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018. pp. 2974–2982. AAAI Press (2018). https://doi.org/10.1609/aaai.v32i1.11794, https://doi.org/10.1609/aaai.v32i1.11794
15. Gronauer, S., Diepold, K.: Multi-agent deep reinforcement learning: a survey. Artif. Intell. Rev. **55**(2), 895–943 (2022). https://doi.org/10.1007/s10462-021-09996-w, https://doi.org/10.1007/s10462-021-09996-w
16. Guo, R., Cheng, L., Li, J., Hahn, P.R., Liu, H.: A survey of learning causality with data: Problems and methods. ACM Comput. Surv. **53**(4) (2020)
17. Heinze-Deml, C., Maathuis, M.H., Meinshausen, N.: Causal structure learning. Annual Review of Statistics and Its Application **5**(1) (2018)
18. Hernandez-Leal, P., Kartal, B., Taylor, M.E.: A very condensed survey and critique of multiagent deep reinforcement learning. In: Seghrouchni, A.E.F., Sukthankar, G., An, B., Yorke-Smith, N. (eds.) Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020. pp. 2146–2148. International Foundation for Autonomous Agents and Multiagent Systems (2020). https://doi.org/10.5555/3398761.3399105, https://dl.acm.org/doi/10.5555/3398761.3399105
19. Heylighen, F.: Stigmergy as a generic mechanism for coordination: definition, varieties and aspects. Vrije Universiteit Brussels, ECCO working paper edn. (2012)
20. Izquierdo-Torres, E.: Collective intelligence in multi-agent robotics: Stigmergy, self-organization and evolution. Brighton BNl (2004)
21. Khan, L.U., Saad, W., Han, Z., Hossain, E., Hong, C.S.: Federated learning for internet of things: Recent advances, taxonomy, and open challenges. IEEE Commun. Surv. Tutorials **23**(3), 1759–1799 (2021). https://doi.org/10.1109/COMST.2021.3090430, https://doi.org/10.1109/COMST.2021.3090430
22. Lalanda, P., Hamon, C., Escoffier, C., Lévèque, T.: iCasa, a development and simulation environment for pervasive home applications. In: IEEE 11th Consumer Communications and Networking Conference (Jan 2014)
23. Leibo, J.Z., Hughes, E., Lanctot, M., Graepel, T.: Autocurricula and the emergence of innovation from social interaction: A manifesto for multi-agent intelligence research. CoRR **abs/1903.00742** (2019), http://arxiv.org/abs/1903.00742
24. Lippi, M., Mariani, S., Martinelli, M., Zambonelli, F.: Individual and collective self-development: Concepts and challenges. In: Proceedings of the 17th Conference on Computer Science and Intelligence Systems, FedCSIS, Sofia, Bulgaria, September 4-7, 2022
25. Lippi, M., Mariani, S., Zambonelli, F.: Developing a "sense of agency" in IoT systems: Preliminary experiments in a smart home scenario. In: 19th IEEE International Conference on Pervasive Computing and Communications Workshops (2021)
26. Mahfoudh, H.B., Serugendo, G.D.M., Naja, N., Abdennadher, N.: Learning-based coordination model for spontaneous self-composition of reliable services in a distributed system. Int. J. Softw. Tools Technol. Transf. **22**(4), 417–436 (2020). https://doi.org/10.1007/s10009-020-00557-0, https://doi.org/10.1007/s10009-020-00557-0

27. Mamei, M., Menezes, R., Tolksdorf, R., Zambonelli, F.: Case studies for self-organization in computer science. Journal of Systems Architecture **52**(8-9) (2006)
28. Mariani, S., Roseti, P., Zambonelli, F.: Multi-agent learning of causal networks in the internet of things. In: Mathieu, P., Dignum, F., Novais, P., De la Prieta, F. (eds.) Advances in Practical Applications of Agents, Multi-Agent Systems, and Cognitive Mimetics. The PAAMS Collection. pp. 163–174. Springer Nature Switzerland, Cham (2023)
29. Oroojlooyjadid, A., Hajinezhad, D.: A review of cooperative multi-agent deep reinforcement learning. CoRR **abs/1908.03963** (2019), http://arxiv.org/abs/1908.03963
30. Panait, L., Luke, S.: Cooperative multi-agent learning: The state of the art. Auton. Agents Multi Agent Syst. **11**(3), 387–434 (2005). https://doi.org/10.1007/s10458-005-2631-2, https://doi.org/10.1007/s10458-005-2631-2
31. Pearl, J.: The seven tools of causal inference, with reflections on machine learning. Commun. ACM **62**(3) (2019)
32. Pearl, J., Mackenzie, D.: The Book of Why. Basic Books, New York (2018)
33. Pinciroli, C., Lee-Brown, A., Beltrame, G.: Buzz: An extensible programming language for self-organizing heterogeneous robot swarms. CoRR **abs/1507.05946** (2015), http://arxiv.org/abs/1507.05946
34. Pollack, J., Bedau, M.A., Husbands, P., Watson, R.A., Ikegami, T.: Learning Ant Foraging Behaviors (2004)
35. Razian, M.R., Fathian, M., Bahsoon, R., Toosi, A.N., Buyya, R.: Service composition in dynamic environments: A systematic review and future directions. J. Syst. Softw. **188**, 111290 (2022). https://doi.org/10.1016/j.jss.2022.111290, https://doi.org/10.1016/j.jss.2022.111290
36. Reid, C.R., Latty, T.: Collective behaviour and swarm intelligence in slime moulds. FEMS Microbiology Reviews **40**(6) (08 2016)
37. Scherrer, N., Bilaniuk, O., Annadani, Y., Goyal, A., Schwab, P., Schölkopf, B., Mozer, M.C., Bengio, Y., Bauer, S., Ke, N.R.: Learning neural causal models with active interventions. CoRR **abs/2109.02429** (2021), https://arxiv.org/abs/2109.02429
38. Schmickl, T., Crailsheim, K.: A navigation algorithm for swarm robotics inspired by slime mold aggregation. In: Swarm Robotics, Second International Workshop, Rome, Italy, September 30-October 1, 2006
39. Schölkopf, B., Locatello, F., Bauer, S., Ke, N.R., Kalchbrenner, N., Goyal, A., Bengio, Y.: Toward causal representation learning. Proc. IEEE **109**(5) (2021)
40. Solar-Lezama, A.: Program Synthesis by Sketching. Ph.D. thesis, USA (2008), aAI3353225
41. Sosic, A., KhudaBukhsh, W.R., Zoubir, A.M., Koeppl, H.: Inverse reinforcement learning in swarm systems. In: Larson, K., Winikoff, M., Das, S., Durfee, E.H. (eds.) Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017. pp. 1413–1421. ACM (2017), http://dl.acm.org/citation.cfm?id=3091320
42. Stone, P., Veloso, M.M.: Multiagent systems: A survey from a machine learning perspective. Auton. Robots **8**(3), 345–383 (2000). https://doi.org/10.1023/A:1008942012299, https://doi.org/10.1023/A:1008942012299
43. Sunder, V., Vig, L., Chatterjee, A., Shroff, G.: Prosocial or selfish? agents with different behaviors for contract negotiation using reinforcement learning. CoRR **abs/1809.07066** (2018), http://arxiv.org/abs/1809.07066
44. Viroli, M., Beal, J., Damiani, F., Audrito, G., Casadei, R., Pianini, D.: From field-based coordination to aggregate computing. In: Serugendo, G.D.M., Loreti, M. (eds.) Coordination Models and Languages - 20th IFIP WG 6.1 International Conference, COORDINATION 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018. Proceedings. Lecture Notes in Computer Science, vol. 10852, pp. 252–279. Springer (2018). https://doi.org/10.1007/978-3-319-92408-3_12, https://doi.org/10.1007/978-3-319-92408-3_12
45. Vowels, M.J., Camgöz, N.C., Bowden, R.: D'ya like dags? A survey on structure learning and causal discovery. CoRR **abs/2103.02582** (2021), https://arxiv.org/abs/2103.02582

46. Wolpert, D.H., Tumer, K.: Optimal payoff functions for members of collectives. Adv. Complex Syst. **4**(2-3), 265–280 (2001). https://doi.org/10.1142/S0219525901000188, https://doi.org/10.1142/S0219525901000188
47. Ye, D., Zhang, M., Vasilakos, A.V.: A survey of self-organization mechanisms in multiagent systems. IEEE Transactions on Systems, Man, and Cybernetics: Systems **47**(3), 441–461 (2017). https://doi.org/10.1109/TSMC.2015.2504350
48. Zaïem, M.S., Bennequin, E.: Learning to communicate in multi-agent reinforcement learning : A review. CoRR **abs/1911.05438** (2019), http://arxiv.org/abs/1911.05438
49. Zambonelli, F., Omicini, A., Anzengruber, B., Castelli, G., Angelis, F.L.D., Serugendo, G.D.M., Dobson, S.A., Fernandez-Marquez, J.L., Ferscha, A., Mamei, M., Mariani, S., Molesini, A., Montagna, S., Nieminen, J., Pianini, D., Risoldi, M., Rosi, A., Stevenson, G., Viroli, M., Ye, J.: Developing pervasive multi-agent systems with nature-inspired coordination. Pervasive Mobile Compututing **17** (2015)
50. Zhang, K., Yang, Z., Basar, T.: Multi-agent reinforcement learning: A selective overview of theories and algorithms. CoRR **abs/1911.10635** (2019), http://arxiv.org/abs/1911.10635
51. Zhang, K., Yang, Z., Basar, T.: Decentralized multi-agent reinforcement learning with networked agents: recent advances. Frontiers Inf. Technol. Electron. Eng. **22**(6), 802–814 (2021). https://doi.org/10.1631/FITEE.1900661, https://doi.org/10.1631/FITEE.1900661
52. Zhu, C., Dastani, M., Wang, S.: A survey of multi-agent reinforcement learning with communication. CoRR **abs/2203.08975** (2022). https://doi.org/10.48550/arXiv.2203.08975, https://doi.org/10.48550/arXiv.2203.08975