Individual and group learning dynamics in evolutionary collective robotics

Nicolas Fontbonne

Sorbonne Université
Institut des Systèmes Intelligents et de Robotique
Laboratoire de recherche en informatique de Sorbonne Université
Ecole Doctorale Informatique, Télécommunications et Electronique (ED130)

Individual and group learning dynamics in evolutionary collective robotics

Pour l'obtention du grade de Docteur ès Sciences par:

Nicolas Fontbonne

Sous la direction de **Nicolas BREDÈCHE** et **Nicolas MAUDET**

Présenté le **14 mars 2023** à l'ISIR, Sorbonne Université, Campus Pierre et Marie Curie devant le jury composé de:

Kagan TUMER	Oregon State University	professeur	rapporteur
Olivier SIMONIN	Université de Lyon, INSA Lyon, INRIA	professeur	rapporteur
Carola DOERR	CNRS, Sorbonne Université	directrice de recherche	examinatrice
Stéphane AIRIAU	Université Paris-Dauphine	maître de conférence	examinateur
Aurélie BEYNIER	Sorbonne Université	maîtresse de conférence	invitée
Nicolas MAUDET	Sorbonne Université	professeur	co-directeur de thèse
Nicolas BREDÈCHE	Sorbonne Université	professeur	co-directeur de thèse

Abstract

With their proliferation in industry and daily life, robots are now increasingly required to interact with each other. This thesis deals with the problem of coordination between robots in a context where they have to learn their control policy autonomously. These policies are optimized with machine learning algorithms that take advantage of a reward function to increase performance incrementally. The structure of this function will significantly influence the learning dynamics and, then, the possible behaviours of the agents.

We first study systems where agents individually receive a local reward adapted to their actions and must converge towards an optimal collective behaviour. We introduce a distributed evolutionary learning algorithm called Horizontal Information Transfert (HIT) that tackles this particular issue. Agents interact on-line in their environment and must learn their control policy with an embedded evolutionary algorithm and a parameter exchange system. It has the advantage of coping with the limited computation and communication capabilities of low-cost robots, which are often used in swarm robotics. We analyze this algorithm's characteristics and learning dynamics on a foraging task.

We then study systems where the reward is given globally to the entire team. Therefore, this evaluation does not necessarily represent each agent's performance, and it can be challenging to calculate an individual contribution. We introduce a centralized cooperative co-evolutionary algorithm (CCEA) that modulates the number of agents' policies modification to find a compromise between evaluation quality and execution speed. This modulation also helps in completing tasks where improving team performance requires multiple agents to update in a synchronized manner. We use a multi-robot resource selection problem and a simulated multi-rover exploration problem to provide experimental validations of the proposed algorithms.

Résumé

Avec leur prolifération dans l'industrie et la vie quotidienne, les robots sont désormais de plus en plus amenés à interagir entre eux. Cette thèse traite du problème de la coordination entre robots dans un contexte où ils doivent apprendre leur politique de contrôle de manière autonome. Ces politiques sont optimisées avec des algorithmes d'apprentissage automatique qui tirent parti d'une fonction de récompense pour augmenter progressivement les performances. La structure de cette fonction va influencer significativement la dynamique d'apprentissage et donc les comportements possibles des agents.

Nous étudions d'abord les systèmes où les agents reçoivent individuellement une récompense locale adaptée à leurs actions et doivent converger vers un comportement collectif optimal. Nous introduisons un algorithme d'apprentissage évolutif distribué appelé Horizontal Information Transfer (HIT) qui s'attaque à ce problème particulier. Les agents interagissent en ligne dans leur environnement et doivent apprendre leur politique de contrôle avec un algorithme évolutif embarqué et un système d'échange de paramètres. Il a l'avantage de faire face aux capacités de calcul et de communication limitées des robots à faible coût, qui sont souvent utilisés dans la robotique en essaim. Nous analysons les caractéristiques et la dynamique d'apprentissage de cet algorithme sur une tâche de recherche de nourriture.

Nous étudions ensuite des systèmes où la récompense est donnée globalement à toute l'équipe. Ainsi, cette évaluation ne représente pas nécessairement la performance de chaque agent et il peut être difficile de calculer une contribution individuelle. Nous introduisons un algorithme co-évolutif coopératif centralisé (CCEA) qui module le nombre de modifications des politiques des agents pour trouver un compromis entre la qualité de l'évaluation et la vitesse d'exécution. Cette modulation aide également à effectuer des tâches où l'amélioration des performances de l'équipe nécessite la mise

à jour de plusieurs agents de manière synchronisée. Nous utilisons un problème de sélection de ressources multi-robot et un problème d'exploration multi-rover simulé pour fournir des validations expérimentales des algorithmes proposés.

Contents

List of Figures

List of Tables

1 Introduction

Individual and group learning dynamics in evolutionary collective robotics.

The natural world is full of examples of self-organizing behaviors. One such example is the phenomenon of synchronous flashing exhibited by fireflies in mangrove forests that occurs without central coordination or external cues. Many collective behaviors observed in the animal kingdom involve more than just synchronization. For instance, ants and bees are known for dividing labor within their colonies, or more complex animals such as hyenas are able to cooperate in hunting. These illustrate the incredible ability of nature to produce self-organizing behaviors at all levels of complexity.

This has inspired engineers and scientists to design autonomous robots. Thus, many so-called multi-robot systems have been created to assist humans in different tasks such as logistics, surveillance, agriculture, etc. This large span of applications has led to a great variety of systems. The robot teams can be composed of three individuals or thousands in a swarm. They can all be identical, or each can have well-defined capabilities and roles. They can have adversarial individual goals or conversely, have aligned objectives and cooperate.**Chapter 2** of this thesis gives an overview of these systems.

In particular, within the field of collective robotics, swarm robotics has recently drawn a lot of attention. This subfield focuses on the design and investigation of simple robotic systems that operate in large groups. These systems exhibit emergent behaviour, displaying remarkable resilience and adaptability through the use of simple individual rules.

Yet, fine-tuning and deploying these behaviours by hand can be challenging. Machine learning methods can be used to overcome this difficulty, but they also have their own limitations. In **Chapter 3**, we emphasize the large span of multi-robot systems and their impacts on the learning schemes. We especially focus on robots' goals, information flows and optimization algorithms.

Each agent goal is described by an objective function that must be optimized to maximize performance. Using machine learning shifts the problem from designing optimal behaviour to designing an effective objective function. If this function is aligned with the system's global objective, the agents' behaviour can ultimately be beneficial to the system. Sometimes, the objective function evaluate the behaviours of the whole team and is not a good representation of one individual contribution. For example, free riders could benefit from the performance of their teammates without contributing to the task. To avoid this kind of detrimental outcome, we can allow the robot to estimate more precisely its contribution to the collective. Numerous methods have been proposed to extract one's contribution or decompose an objective function for more tailored assessments.

Another critical factor in the design of algorithms is the information flows between agents. Agents can learn more efficiently and benefit from all experiences and explorations when they can share information on a communication network. Centralizing the information also allows methods to mitigate learning stability and calculate better individual contributions.

Goals and information flow will then condition the type of optimization algorithm we can expect to use. Indeed, recent years have led to great developments in deep reinforcement learning algorithms. However, they usually require a large amount of information flow during the learning phase, which may not be available in practice. On the other hand, more lightweight methods exist in the realm of evolutionary algorithms. This algorithm class takes inspiration from the natural evolution of living organisms to optimize the behaviour of robots.

This thesis aims to design and analyze multi-robot evolutionary algorithms with two different goals definition and information flow setups. We explore the following questions:

- how to deploy and optimize a swarm of robots in an unknown environment?

- how to estimate the marginal contribution and stabilize learning in a setup where robots interact in an episodic manner and replay is costly?

More specifically, the first setup concerns robots deployed in their environment, ready to solve a given task. They can share messages and have individual goals. A class of optimization algorithms inspired by the evolution of living organisms is being used to optimize the behaviour of robots. In particular, the process of horizontal gene transfer, where an organism can integrate genetic material from another organism that is not its ancestor, is being studied. This mechanism is commonly found in bacteria and is being used as the basis for a new online optimization algorithm, introduced in **Chapter 4**. The challenge it addresses is coping with low-cost robots' limited computation and communication capabilities. In order to do so, the algorithm decouples computation and communication. It ensures the learning of efficient control policies even when only a limited amount of information can be exchanged between neighbouring robots.

But learning after the deployment of the robots in the environment can be difficult. A simplification is generally to learn by episode, that is, by repeating the same task in a loop to sample some experience while trying to improve between attempts. This context allows for centralizing information between agents and synchronizes their learning.

In **Chapter 5 and 6**, we introduce a new cooperative co-evolutionary algorithm to teach a team of robots to spread on a set of resources or some points of interest. We assume that only the performance at the team level can be accessed. In this situation, the reward signal expressing the agents' goal is a poor representation of their individual performances. In order to accurately capture the contribution of agents and stabilize the learning process, we propose to modulate the number of agents, which is updated in-between team evaluations. This allows finding a compromise between accurately estimating the marginal contribution of agents while maintaining the possibility of modifying multiple agents simultaneously. The latter is critical in cases where the task requires coordination between agents.

2 Multi-Robots Systems

Robots are more and more present to assist humans in their daily life and industrial work. Economists are speaking of the fourth industrial revolution (4IR) of robotic and automation technologies (Schwab, 2017). Machines are of great use in assisting humans with complex and repetitive tasks.

Given this increase, robots can no longer act in isolation. Instead, they are necessarily brought to interact with each other physically in their environment or by message via communication channels. The study of multi-robot systems started in the 80s in many different settings and environments. They are now used in real-world situations in warehouses and are planned to be used in many other cases.

The goal of this chapter is to discuss the *what*, *why* and *how* of multi-robots systems. It first details a taxonomy of multi-robot systems by classifying them on how they interact. It then describes the types of behaviours that robots are led to perform and the real-life application they could help achieve. Moreover, it presents different design methods to create performing teams.

2.1 Multi-Robot Systems

The development of the human species owes much to its ability to cooperate (Hamilton, 1963, 1964; Pennisi, 2005). Indeed, many problems can benefit from the collaboration of several agents and do not have to be solved sequentially. This idea of modularization (Arkin et al., 1998) is noticeable, for instance, in many construction processes where many parts and elements can be built independently and then assem-

bled together. Thanks to this division of labour, one can expect overall performance to be better than the sum of individual performances. Using several robots can also be helpful to guarantee robustness and redundancy in a team to avoid depending on the good functioning of a single robot.

When several robots are brought to interact together, we speak of a *multi-robot system* (MRS). It can range from a simple group of sensors acquiring and processing data to a set of very complex moving robots. In this thesis, we are interested in the former: each autonomous unit has its own set of sensors and actuators and interacts explicitly or implicitly to perform a given task. While the goal is the same for everyone, several different robotic platforms can be used in the same team, in which case, we speak of a *heterogeneous team.* On the contrary, when all robots are identical, we speak of a *homogeneous team.*

We also define a large team of homogeneous relatively simple robots as a *swarm.* Beni, 2004 specifies that the size of a swarm should be neither so large that it is necessary to use statistical averages to analyze it nor so small that it should be treated as a few-body problem. Swarm robotic is based on the principle of *superadditivity* (Parker, 2008), where the whole result (collective behaviour) is better than a simple sum of all its parts (the agents' behaviours).

Many classifications have been proposed over the years to sort MRS. Previous reviews have classified MRS by their level of autonomy (Rizk et al., 2019), the type of interaction between the robots and the environment (Farinelli et al., 2004), the relationship between the robots' goals (Parker, 2007), or the applications and associated behaviours (Brambilla et al., 2013; Murray, 2007). In this section, we detail the different architectures of interactions between robots. The following section then highlights the behaviours and tasks that robots must solve before describing how they apply to actual applications.

Figure 2.1 represents some typical relationships between robots. Robots can ignore each other or be aware of others' presence. They can communicate with each other symmetrically or hierarchically. They also can interact via a centralized authority to coordinate their behaviours with global information out of their reach.

- *Un-aware robots:* When robots are not aware of the presence of others, cooperation can emerge in a goal-oriented way. In this case, we usually refer to *collective*

Figure 2.1: Simplified representation of different multi-robotics setups. (a) Unaware robots can only sense their teammate's impact by observing their environmental effects. (b) aware robots use their sensors to monitor others' actions directly. (c) networked robots can share information through more or less long-distance communication channels. (d) in hierarchical systems, some robots have control over the action of others through a communication channel. (e) in centralized systems, robots share information with a central authority that makes global decisions for them.

robotics. Robots may be able to observe the impact of others through their effects on the environment. For example, in a foraging task where all robots must carry items around, one can sense the impact of a teammate by observing how the items to be picked up are redistributed. This type of implicit communication through the world is often called *stigmergy* (Grasse, 1959; Kube and Bonabeau, 2000). It has the advantage of not depending on costly communication channels and protocols.

- *Aware robots without communication*: in this case robots reason about the actions and intentions of their teammates without relying on a direct communication canal. They can use sensors to directly observe their actions (Huber and Durfee, 1995) and plan accordingly.
- *Networked robots*: Explicit communication between robots can be used to share information, synchronize actions and negotiate to influence the decision-making of other nearby robots. There are many ways in which robots or agents can communicate with each other, but issues of frequency, bandwidth, message collision, and locality of communication come into play.
- *Centralized system*: MRS can also be characterized by the presence of a centralized software that coordinates other robots. The *controller* aggregates data from the whole team and can make informed decisions, impossible to make at the local level. But it may make the system vulnerable to a single point of failure. Communicating all information to a central location at a frequency suitable for real-time control can also be challenging.

- *Hierarchical system*: Different levels of hierarchy can also modulate the level of distributions of sensing and decision-making. For example, some robots can act as leader nodes and asymmetrically share decisions and information with other teammates.
- *Human-robot interaction*: Multi-robot systems can also benefit from interaction with human operators who guide the team's decision making between robots (Cummings, 2004; Kolling et al., 2013). This allows autonomy to be delegated to a human with a more global view of the task, its state of completion, and the trade-offs to be made.

Each of these architectures has its limitations, advantages, and disadvantages. Choosing one will mainly depend on the available robotic platforms, communication technologies and the desired behaviour. The following section describes the types of collective behaviours used in multi-robot systems.

2.1.1 Collective behaviours

The completion of a task by a team of robots usually involves certain steps. Depending on the architecture of the multi-robot system in place, these steps can be explicitly or implicitly solved by assembling identified building blocks called *collective behaviours*. Brambilla et al., 2013 classify these collective behaviours into four categories: *collective decision-making*, *spatial-organization*, *navigation* and any other collective behaviours.

Collective decision-making

Collective decision-making behaviours are any form of decision construction requiring information or preferences of multiple robots. It can be to reach an agreement on a variable when forming a consensus, or to form a specialization for each robot when assigning tasks.

We speak of *consensus formation* (Olfati-Saber et al., 2007) when the robots form a consensus on continuous or discrete variables. Continuous consensus achievement has been extensively studied in the context of flocking, where robots must agree on a certain motion direction (Olfati-Saber, 2006). Continuous consensus can also be used to perform collective perception (Valentini, Brambilla, et al., 2016), collective

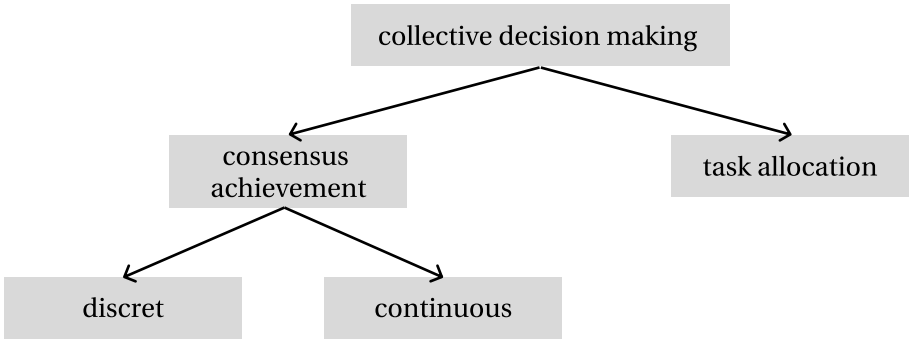


Figure 2.2: Classification of collective decision-making reproduced from Valentini et al., 2017

mapping (Aragues et al., 2012) or distributed signal processing and is widely used for sensor networks.

For discrete consensus formation, a team of robots must agree on a specific option in a set of choices. For example, a team of robots may have a predefined set of strategies and must agree on which strategy to execute (Bowling et al., 2004; Kok, Vlassis, et al., 2003). Valentini et al., 2017 have coined the consensus on N possible choice as the *best-of-n problem* and many decision-making methods have been proposed in the swarm community, using urn models, voter models or majority rule. Blockchain technology also offers a promising method for decentralized decision-making for strongly heterogeneous systems (Castelló Ferrer, 2018).

Figure 2.3 describes a typical collective specialization workflow (Rizk et al., 2019) used by teams of robots. To solve a complex task, teams of robots can divide it into smaller, simpler tasks, and can leverage their own skills to solve them. These are the multi-robot *task decomposition*, *coalition formation* and *task allocation problem* (MRTA). Task decomposition is usually the first step of decision making for team of robot. The feasibility of such a decomposition is very specific to the application domain and the robot team that has to solve it (for example, Chen et al., 2010 in the soccer domain). When the problem has been decomposed, it is a matter of dividing the robots into groups and assigning them a task for which they will be responsible. These are the problem of *Multi-robot coalition formation* (MRCF) and *Multi-robot task allocation* (MRTA) (Gerkey and Matarić, 2004).

There exists a large set of techniques for addressing multi-robot task allocation prob-

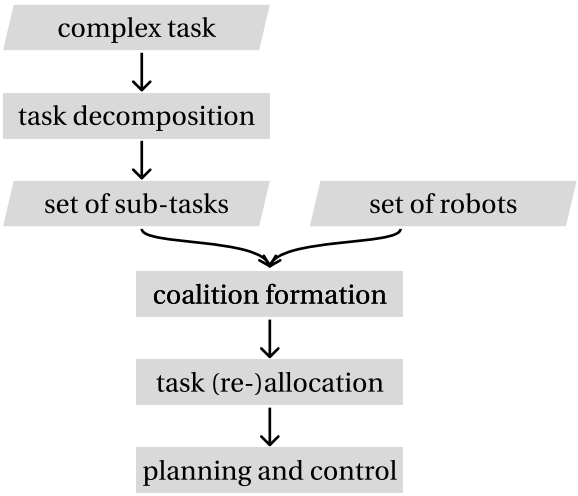
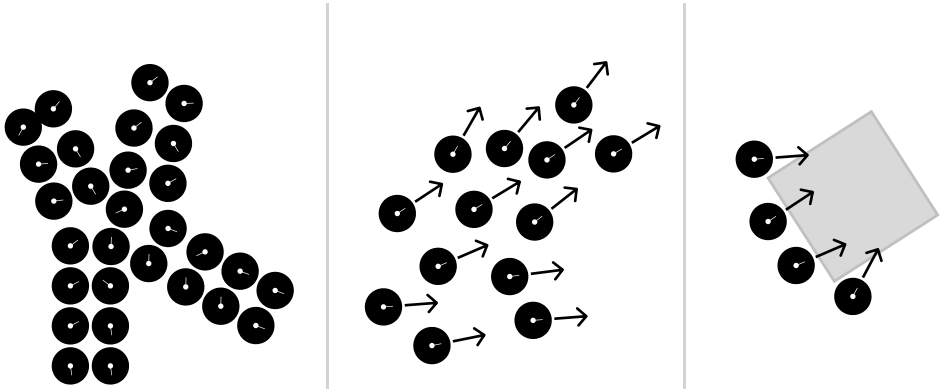


Figure 2.3: Typical collective specialization workflow of a multi-robot system. The first steps of task decomposition, coalition formation, and task allocation are regularly performed by a central authority, that is either a central computer or a human. Planning and control are usually done independently by each robots.

lems (MRTA) (Khamis et al., 2015), for example, using auction in market-based approaches (Zlot and Stentz, 2006). Decentralized task allocation can be performed using a consensus-based auction algorithm (CBAA) (Choi et al., 2009). It works by alternating phases of auctions and consensus until convergence. To tackle the possible physical proximity of tasks, the consensus-based bundle algorithm (CBBA) allows grouping assignments into bundles for bidding. Furthermore, the consensus-based grouping algorithm (CBGA) (S. Hunt et al., 2014) allows the grouping of robots for tasks that would require heterogeneous robots collaboration.

Collective spatial-organization

Spatial organizations are behaviours that focus on how to organize and distribute robots and objects in space. They concern aggregation (Garnier et al., 2005; Schmickl and Hamann, 2011; Soysal and Şahin, 2006; Trianni et al., 2003), chain formation (Nouyan et al., 2008) or, more generally, pattern formation (Rubenstein et al., 2014). These behaviours are powerful building blocks that allow easier message exchanges or prepare for a more complex task. For example, Guerrero-Bonilla et al., 2017 showed that formation creation could enable resilience, consensus achievement and cooperation even in the presence of malicious or malfunctioning robots.



(a) pattern formation (b) flocking (c) object transportation

Figure 2.4: Illustrations of some multi-robot cooperative behaviours. (a) pattern formation involve robots assembling themselves to build a predefined shape (Rubenstein et al., 2014). (b) Flocking concern any team of robot that moves in a direction and where all robot adapt their velocity based on their neighbours. (c) Object transportation is used when a team of robot must move objects too heavy for an individual.

Robots spatial organization can also be inspired by nature, particularly by reaction-diffusion systems studied in developmental biology. This is the concept of morphogenesis. Using it, Slavkov et al., 2018 have demonstrated good performance of self-assembling with hundred robots.

We can finally think of the self-assembly of robots for construction processes or to guarantee more stability during navigation (Groß and Dorigo, 2008). In *reconfigurable modular robotics*, the idea is pushed forward by designing autonomous robot modules that can physically connect into a larger ensemble. This allows the robot ensemble to have self-repairing capabilities, and individual robot-modules can share energy and establish communication buses.

Collective navigation

Navigation behaviours are the coordinated movement of a team of robots in the environment.

This can be a simple coordinated motion in arbitrary form like flocking or a movement in formation (Olfati-Saber, 2006). Such methods, inspired by physics, allow the exploration of hard-to-access forest-like environments (Soria et al., 2021).

Collective navigation can also concern more goal-oriented motion where robots must navigate the environment to explore it (Juliá et al., 2012). In collective mapping, multi-robot teams must create an environment map. The exploration can be seen as task allocation problems where robots must allocate themselves to the boundaries of unexplored portions of the environment (Bautin et al., 2012; Yamauchi, 1997, 1998).

If robots need to collect information along the way, we speak of multi-robot informative path planning (Meliou et al., 2007; Singh et al., 2009). Its goal is to compute a path for each robot that would maximize the accuracy of the measurement estimate under constraints (e.g. small battery life). For example, if the robots move in a space-time field, there are usually correlations between closest points (Krause et al., 2008), which makes some measures more or less useful, and can lead to efficient path planning.

We can also consider robot patrolling in an environment by visiting some points of interest as regularly as possible (Glad et al., 2008).

Some robots can also carry objects and may need coordination to move objects too heavy for an individual (Groβ and Dorigo, 2008). Collective object transportation can be achieved through stigmergy: the action of a robot in collective transport modifies the stimuli perceived by other robots and, in turn, produces changes in their actions (Bonabeau et al., 1999). Different approaches rely on leader-follower principles, where a leader plans the trajectory and controls motion while the remaining robots coordinate to support the leader's movement (Teh et al., 2020; Wang and Schwager, 2016a, 2016b). Finally, in more centralized settings, an external supervisor can observe the collective motion and sends appropriate control actions to the robots to guide the object toward a target direction (Shahrokhi and Becker, 2016). Recently, collaborative drone transportation has also shown promising results (Horyna et al., 2021; Mellinger et al., 2013; Tagliabue et al., 2019).

Summary

These building blocks can then be composed to solve more complex tasks. In the next section, we briefly describe real-world applications of multi-robots and swarm systems that use these collective behaviours to assist humans in various tasks.

2.1.2 Applications

This section reviews several real-world applications where teams of robots can be helpful. Some multi-robot systems are already deployed and used in the industry, notably for warehouse management and environmental monitoring. In the near future, fleets of robots could more intensivelly be used for fire-sensitive forest monitoring, precision agriculture, or automated delivery of goods. More prospective applications concern the cheap and fast construction of buildings in case of emergency or disaster. Currently, applications are still limited to usage in a safe, controlled environment. But future deployments in uncontrolled environments are promising.

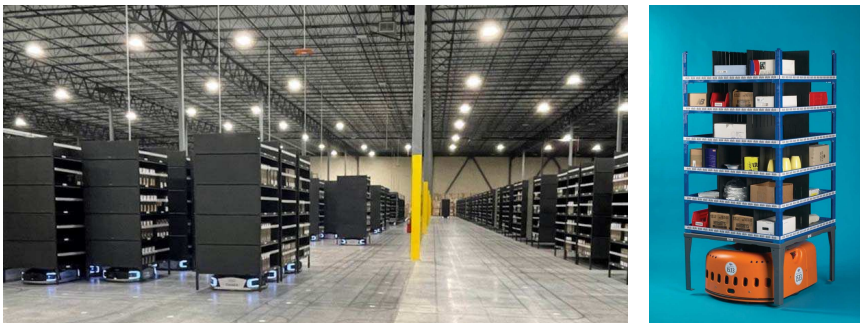
This section reviews a number of these industrial applications and the research approaches around these issues. It highlights the degree of centralization or decentralization and the collective behaviours building block used in the various problems and solutions.

Logistics

Currently, the leading industrial application for multi-robots is warehouse management. In non-automated warehouses, human workers have to move to retrieve orders from shelves. But it is a tedious work that can involve health risks when transporting goods. Nowadays, multiple automation techniques have been proposed to allow more efficient storage and retrieval of goods (Boysen et al., 2019).

Robotic Mobile Fulfillment Systems (RMFS) uses robots to move shelves to the workers in an optimal manner. The company *Kiva Systems*, now known as *Amazon robotics*, has developed a robot that slips into a row of shelves (see Figure 2.5.b), lifts them and brings them the worker (Mountz et al., 2008). Figure 2.5.a represents a warehouse arranged by the Geek+™ company to optimize their customer delivery.

Figure 2.6 represents a typical simple warehouse with a robotic mobile fulfillment system. All of these systems work similarly. Hundred of mobile robots carry selves from a storage area to a picking station for human operators. In these configurations, robots are often called *automated guided vehicle* (AGV) as they are kept constrained to guided lines (either visually or using radio waves, magnets or laser). This allows humans to retrieve stored items without moving around the storage area, increasing



(a) (b)

Figure 2.5: Exemple of robots used in warehouses for RMFS. (a) Warehouse using the Geek+™ solution and (b) Kiva robot carrying an inventory pod (source)

productivity and reducing accidents.

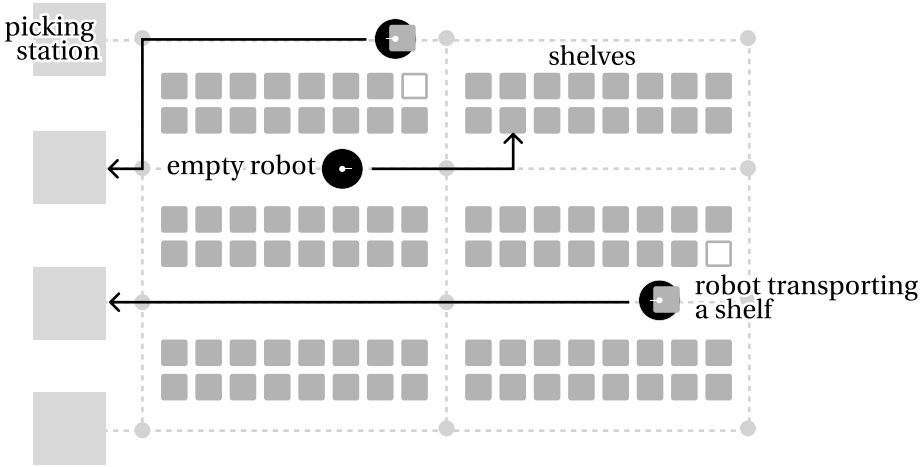


Figure 2.6: Warehouse with multi-robot logistic. Robots must spread themselves between the shelves. They independently plan their trajectories to get the pods and bring them to the human stations. Figure reproduced from Wurman et al., 2008.

The warehouse is ideal for applications with many robots because the controlled environment has no unexpected events, and stable communications can be guaranteed. A global planner can therefore organize and coordinate all robots (Wurman et al., 2008). Many optimization problems are involved in such a system (Enright and Wurman, 2011). In particular, the collective behaviour of task allocation is solved by a centralized *job manager*. This centralization will take customer orders and assign them to robots, which will have to bring them to a station to be picked up by a human worker. It is a question of keeping the operators busy to maximize productivity while limiting the number of robots to limit costs. Path planning to the order's storage and collision

avoidance are handled by each robot individually and requires local coordination with neighbours.

After the warehouse, the goods have to be brought to the individual and shops. With the emergence of autonomous vehicles, this task is about to be automated. The problem of distributing tasks on a fleet of vehicles is the well-known *travelling salesperson problem* (TSP). Drone delivery to individuals has shown some promising results for transporting small parcels, food, and medical supplies (Benarbia and Kyamakya, 2021; Kwak et al., 2022; Rosser Jr et al., 2018). With warehouse management, these technologies can impact how we store goods and distribute them.

Monitoring

Many monitoring applications today make use of sensor networks (Kandris et al., 2020). Wireless sensor nodes are a group of spatially distributed sensor nodes connected by wireless communication (Akyildiz et al., 2002). However, sometimes, sensors cannot be left in place, or the object of analysis is too dangerous to place them initially. Robots are then particularly suitable for monitoring hazardous and hard-to-access locations.

Indeed, robots have been used to monitor animals in their habitats (a group of birds in Cliff et al., 2015, and fishes in Tokekar et al., 2013) and for scientific data measurement (wind in Neumann et al., 2012, soil in Tokekar et al., 2016 and water temperature in Duarte et al., 2016). More dangerous situations can include monitoring accident areas (Maza et al., 2011), hazardous materials (e.g. bacteria in Silva et al., 2012, chemical source in Ferri et al., 2011), high voltage line (Uzakov et al., 2020) or controlling fire sensitive areas (Roldán-Gómez et al., 2021).

Motoring large areas usually involve spatial organization behaviours and coordinated and informative path planning to spread on large areas and maximize the amount and quality of data collected by robots.

Space monitoring has also been used by the European Space Agency with Cluster II, a group of four satellites that collects data on the impact of solar activity on the Earth's magnetosphere (C. P. Escoubet, 2000; C. Escoubet et al., 1997). The four satellites fly in a tetrahedron formation and are thus able to measure data in 3D (Dow et al., 2004). They must regularly perform joint manoeuvres to comply with requests from the Joint

Science Operations Centre cluster.

Agriculture

The agricultural sector had reached a significant turning point. It is one of the sectors that emit the most greenhouse gases (Hannah Ritchie and Rosado, 2020) and will be the hardest impacted by climate change. Over the years, many approaches have been proposed to produce food while limiting its impact on climate, land use and biodiversity.

The use of robots could be a solution to some of these problems. *Precision agriculture* (Bechar and Vigneault, 2016; Cheein and Carelli, 2013; García-Pérez et al., 2008; C. Zhang et al., 2016) allows farmers to use data science to make informed and accurate decisions on every plot of land. In particular, multi-robot systems can be used to collect data automatically and apply solutions in the field.

Multi-robot precision agriculture usually involves a *heterogeneous* team of unmanned aerial vehicles (UAVs) that inspect terrains and assess affected areas and unmanned ground vehicles (UGVs) that use this collected information to efficiently and accurately perform crop treatments. Depending on the proposed solutions, the degree of centralization can vary. In many cases, these systems require human intervention, and a lot of research is being done on the development of robot-machine interface platforms.

Figure 2.7: Fleet of drone flying above a field to gather data and detect the distribution of crops such as in Lottes et al., 2017

More precisely, UAVs plan trajectories to ensure good measurement coverage while limiting the amount of energy used for travel. The challenge of *multi-robot information gathering* is, therefore, to plan complex trips and carry out measurements such as crop ripening stage, soil humidity, fertilizer levels, the presence of pets, weeds

and weather damage. For example, figure 2.7 illustrates a typical crop distribution detection performed in Lottes et al., 2017. Communication about this data must also be secured to prevent strategic data leakage (Dutta et al., 2021).

After this data collection, UGVs use this information to *allocate tasks* between the different robots and perform the required action such as planting seeds and applying pesticides. Most harvesting operations require dealing with natural objects, such as fruits and leaves, that have high variability in shape, texture, colour, size, orientation and position. Therefore, creating a reliable robotics platform is still challenging, and many researchers and companies are tackling the problem.

Search and rescue

Beyond monitoring, multi-robot can also intervene in *search and rescue* (SAR) operations (Grayson, 2014; Grogan et al., 2018; Hayat et al., 2016; Queralta et al., 2020; Shakhatreh et al., 2019). These are very popular problems and have benefited from many international collaborations (Cubber et al., 2017; De Greeff et al., 2018; Kruijff et al., 2014; Ollero et al., 2005) and competitions (Kitano et al., 1999).

Multi-robot systems can cover more ground and search more effectively than a single robot. They can work together to search a larger area, and can divide the search area into smaller sections to make the search more efficient. Additionally, having multiple robots can provide redundancy, so if one robot becomes damaged or fails, the other robots can continue the search. This can be particularly useful in difficult or hazardous environments, where a single robot might be unable to operate effectively. Robots can also be used by firefighters to help extinguish large fires (Naghsh et al., 2008; Roldán-Gómez et al., 2021).

Multi-robots for SAR are typically semi-autonomous systems with varying amounts of human supervision (Kolling et al., 2013). For example, the human can intervene to allocate tasks and mission to robots (Maza et al., 2011). We speak of *shared autonomy* when robots autonomously control the majority of their degrees of freedom, while a control interface allows for a human operator to control a reduced number of parameters defining the global behaviour of the system (Marion et al., 2017). For example, to control the path shape of the robot team using a reduced amount of freedom and provide real-time feedback on the performance (Masone et al., 2014).

Health care

Many public health issues are understaffed or present a safety risk to the worker. Multi-robots systems can be used to assist with task such as dispensing medication and transporting medical equipment. This could help to improve the efficiency and effectiveness of health care delivery (Ackerman, 2011; Rosser Jr et al., 2018).

In the fight against infectious diseases, teams of robots are a promising technology for many tasks of spraying, disinfecting, cleaning, treating, detecting high body temperature/masking. Alsamhi and Lee, 2020 propose for that a framework of collaboration between all robots of a strongly heterogeneous team and use blockchain technology as a consensus method for robot coordination.

Another understaffed health issue is caring for people with reduced mobility and elderly people. Due to the increase in longevity and the decrease in the birth rate, it is becoming more and more challenging to meet the need for healthcare personnel to support elderly people living at home or in nursing homes. Barber et al., 2022 proposes using a heterogeneous team of robots to assist elderly people living alone. This team comprises a monitoring robot that measures their well-being and a manipulator robot that can help them in their daily tasks. In addition to these two robots, the system benefits from the help of a set of low-cost external sensors, a medical monitoring bracelet and an Android application that form an automated home environment (AAL).

Construction

In nature, animals have shown incredible construction abilities to build nests, protection barriers or traps (Hansell, 2007). The diversity of these constructions comes from the diversity of the groups of animals that build them, their methods of collaboration and the size of their teams. Inspired by the animal kingdom, construction could also be performed by a multi-robot system. Indeed, there is a strong demand for safe, durable, and affordable housing.

Collective robotic construction (CRC) (Petersen et al., 2019) could be one answer. It involves multiple robots coordinating to build a structure more extensive than a single robot. The robot teams will vary widely depending on the materials used for construction. These can be divided into two categories: discrete materials such as

bricks, spacers or sandbags, and continuous materials such as concrete, foam and various types of fibre.

Thus, the degree of centralization will significantly impact the performance and simplicity of the perception and decision-making problems. We can sort the different approaches according to their use of centralized, hybrid, or decentralized perception and decision-making. The centralized approach is the most common as it allows external monitoring of construction progress and dispatching individuals to assembly sites. For example, Augugliaro et al., 2014 uses four drones to assemble a 6-m-tall tower composed of 1500 foam modules. A central controller performs the path planning. The decentralized approach extends the concept of stigmergy by allowing robots to store information in the environment for their teammates. (Wawerla et al., 2002; Werfel and Nagpal, 2006). In between, hybrid approaches often involve a central controller that maintains information about the global structure and the positions of the robots. This controller may specify assembly locations (Yun and Rus, 2007) or regions (Yun et al., 2011), and communicate this information to the collective group of robots.

Just like animals, robot teams need feedback on their construction and must therefore overcome the problem of their local perception. It is also a matter of dividing the tasks and having a cooperation mechanism that allows for building a stable structure. Unfortunately, robotic platforms allowing such features do not exist yet, but many teams are working on different methods.

Art and entertainment

Swarm robotics has the potential to be used in a variety of applications in art and entertainment.

One of the most impressive artistic applications of the use of multi-robot systems is drone shows (Waibel et al., 2017). Swarm of aerial robots are used as part of a live performance, such as a dance or music show and are programmed to move in ways that create interesting visual or auditory effects (Alonso-Mora et al., 2012). In these performances, drones navigate autonomously, piloting themselves following choreographed and pre-programmed motions while supervised by a human operator.

Swarm robots can also be used to create interactive installations, such as sculptures,

exhibits or toys that respond to the presence and movements of humans (**glowbots**). For example, a swarm of robots could be arranged in a specific pattern, and as visitors approach, the robots could change their behaviours, movement or LEDs in response (Alhafnawi et al., 2020). It questions the perception of the swarm as a whole behavioural entity (Dietz et al., 2017) and the establishment of a possible affective relationship with it (Santos and Egerstedt, 2021). Teams of robots have been used for interactive painting on a canvas (Santos et al., 2020).

Robocup

On the research side, robot soccer has been identified as a benchmark for multi-robot research. To stimulate research, the RoboCup (Kitano, Asada, Kuniyoshi, Noda, and Osawa, 1997; Kitano, Asada, Kuniyoshi, Noda, Osawa, and Matsubara, 1997) has proposed a highly competitive multi-robot soccer tournament. In all the leagues of the event, the robotic platform is fixed, and the players must program the control systems of each robot and the team strategy. In the *middle size league*, all robots have their sensor on board, and they must deal with uncertainty and reconstruct global information about the environment. On the other hand, the *small size league*, a top view camera, gives global information to all robots, and coordination can be centralized. As individual performances are well known, the differences in level between the teams are now usually based on the ability to coordinate.

Summary

With all these applications, multi-robot systems are very diverse. Table 2.1 summarizes their characteristics. The team composition details the type of robots used to solve the task. The architecture column highlights the type of communication between robots and a possible centralization. Finally, the collective behaviour column shows the types of behaviour to be performed and composed to solve the task.

As robots begin to be deployed in the real world, there is a need to ensure that their use is safe and benefits the common good (Veruggio et al., 2016). E. R. Hunt and Hauert, 2020 propose a checklist of questions to ask before taking robots out of the lab or warehouse. In particular, it is about asking questions about the ethical alignment of the application. Applications must be used for social good and comply with all relevant laws and regulations for the domain(s) of deployment. Many international

Application	**Team composition**	**Information flow**	**Collective behaviours**
warehouse management	homogeneous (ugv)	centralized	task allocation
monitoring	homogeneous	centralized and networked	area coverage, formation control
agriculture	heterogeneous (ugv, uav)	human-robot, hybrid	path planning and task assignment
search and rescue	homogeneous (uav)	human-robot interaction and networked	task allocation, area coverage, formation control and pattern formation
elderly care	heterogeneous	networked	task allocation
health	heterogeneous	centralized	consensus formation
construction	homogeneous	centralized, hierarchical or networked	self-assembly, cluster formation and collective transport
drone shows	homogeneous	centralized	self-assembly, collective transport

Table 2.1: Summary of real-life multi-robot applications. The team composition column is the type of robots used to address the task. The architecture column highlights the type of communication between robots and a possible centralization. The collective behaviour column shows the type of behaviours performed and composed to solve the task

actors are committed to preventing or limiting the military use of autonomous robots (Asaro, 2012), such as the *Stop Killer Robots* movement (Gubrud, 2014).

Current applications still only concern well-controlled environments, such as a warehouse. However, safety still needs to be significantly improved (Evans, 2020). Deployment in an uncontrolled environment requires essential safety procedures not to hurt humans and human activities. Users must be able to interact with the team of robots to prevent physical harm caused by the individuals and their behaviours. Given the environmental constraints, it is also essential to consider the costs and benefits of these technologies and the social and economic paths they may imply.

2.1.3 Design approaches

All of these applications have quite diverse robot and communication configurations. This plurality has favoured the emergence of numerous approaches for designing and analyzing these multi-agent systems (Shoham and Leyton-Brown, 2008a). So far we have presented a general overview of the properties and choices that come with designing a MRS. Namely, we discussed the *what* and *why* of multi-robot systems. In this section, we focus on the *how*.

We distinguish between the so-called manual approaches, which rely on the designer's ingenuity to create an adequate behaviour for each robot, and the automatic methods, which will allow the robots to adapt their behaviour through their repeated interactions with the environment.

Manual design methods

Real-world applications mainly use manual design because safety requires predictable and straightforward behaviours. Over the years, many approaches have been created to program robots and organize their collaboration (Parker et al., 2016; Siciliano et al., 2008). This section highlights three design approaches: *reactive approaches*, *behaviour-based approaches* and *deliberative approaches*.

- **Reactive approaches:**
  The simplest robots do not plan for future actions and are only reactive to their sensory inputs (Braitenberg, 1984; Brooks, 1986). The architecture of reactive systems is not based on reasoning or planning but rather on a direct connection between sensors and effectors. This is inspired by the biological concept of *stimulus-response*. Reactive systems are usually made up of a programmed set of rules that, when given sensory inputs, produce the desired output actions. This allows reactive systems to compute very quickly and be useful in situations where quick reactions are necessary. However, because these systems do not keep a representation of the world or store information, they are limited in their ability to deal with uncertainty and novel situations.
  In the multi-robot case, reactive systems often rely on the concept of emergence, where complex behaviours arise from the collective behaviour of agents with simple individual capacities. This means designing reactive distributed systems

is a bottom-up process involving iteratively crafting individual behaviours and observing the collective results.

A popular method of reactive robotics is inspired by physics and considers each robot as a particle. The robots interact with each other thanks to social potential fields (Reif and Wang, 1999) and are attracted by the goal that exerts an attractive force on them. This framework is very useful for creating pattern formation behaviour (Spears et al., 2004), collective exploration or coordinated flocking movements for disaster relief (Reynaud and Guérin-Lassous, 2016).

- **Behaviour-based approaches:**
  After the development of reactive approaches, behaviour-based approaches (Arkin et al., 1998) were proposed in an effort to improve upon their ability to respond to changing environments. This was mostly popularized by the subsumption architecture proposed by Brooks, 1986. In behaviour-based architectures, the robot control is constituted of several basic behaviours, which are organized in separate modules. The most common method used to describe robots behaviours manually is *probabilistic finite state machine* (PFSM) (Minsky, 1967). By allowing interaction of influence with each other in these finite state machines, very complex behaviours can be achieved. These types of design are easy to set up and allows complex behaviours of task allocation (Valentini, Ferrante, Hamann, et al., 2016b), aggregation (Schmickl and Hamann, 2011) and chain formation (Nouyan et al., 2008). The global system can also be modelled as a Petri Net that generalize state machine for concurrent execution (Costelha and Lima, 2012; McCarragher, 1993; Peterson, 1977).
  In the ALLIANCE (Parker, 1998) framework, the robots have a set of high-level behaviours and can influence their teammate on the choice of behaviours by influencing a motivation level.
  Matarić, 1995 proposed the Nerd Herd, a group of 20 identical robots that were only able to detect obstacles and other robots. Each robot had a set of pre-programmed behaviours, such as obstacle avoidance, homing, aggregation, dispersion, following, and safe wandering. By combining these behaviours, the system was able to achieve higher levels of functionality. For example, the robots were able to engage in collective foraging by using a temporal combination operator to switch between avoidance, dispersion, following, homing, and wandering.
- **Deliberative approaches:**

This approach, also known as the Sense-Model-Plan-Act architecture (Albus, 1991; Iocchi et al., 2000; Matarić, 1998), has been a classical approach in robotics (Nilsson et al., 1984) and AI for representing high reasoning capacities. It is based on the idea that sensory information is processed using the robot's internal knowledge in order to plan and determine the next action, and as such relies on an internal representation of the world. However, this approach can be computationally expensive and lacks real-time reactivity.
In the case of multi-robot systems, an additional level of global control known as social deliberative approach (Iocchi et al., 2000) can be defined. In a social deliberative multi-robot system, a global strategy is planned to allow the system as a whole to respond to changes in the environment (e.g. task re-allocation). For example, in Werger and Matarić, 2000, the robots were individually behavior-based, but group-level deliberation was achieved through an architecture called *Broadcast of Local Eligibility*. Each robot could evaluate its ability to perform a given task and then broadcast that value. The robot with the highest value would then claim the task, allowing for efficient task assignment in the system.

Automatic design methods

However, the dynamic nature of the environment and the tasks that the robots have to address require robustness and adaptability. Also, manually programming robots independently to perform grouped tasks is not necessarily intuitive from the designer's point of view, who typically uses a *code-and-fix* approach (Brambilla, Pinciroli, et al., 2012). In this context, using *automatic design methods* is critical for multi-robot systems and can help to reduce the effort of the developers in designing a policy than achieve a particular collective behaviour.

The learning method and its temporality will largely depend on the available hardware. As seen previously, multi-robot systems can range from simple opposition of small teams in games like soccer to a whole crowd of independent robots or a swarm of robots with limited communication and computation capabilities. On the basis of these characteristics, many methods can be borrowed from the field of *multi-agent learning* (MAL).

Machine learning has allowed a significant development of adaptive methods. However, learning in multi-agent systems went back a long time and started the emergence

of game theory. As a result, many parallel and complementary approaches have been developed to allow and analyze the adaptation of agents to their tasks, environment and peers. According to Tuyls and Stone, 2017, multi-agent learning is defined as one or more of the autonomous entities improving automatically through experience. It regroups three classes :

- *Individual learning*, when a single agent learn in interaction with multiple non-learning agents
- *Population learning*, when multiple agents learn together in interaction
- *Protocol learning*, when only the system of interaction between agents is learned (Parkes, 2004). None of the agents can modify their behaviours, but the designer can control the mechanism of their interaction. For example, an auction house cannot modify the bidder's behaviour but may adapt the bidding rules (e.g. English auction, Vickrey auction, Dutch auction, etc.).

The next chapter will dive deep into the individual and population learning that are extensively used for optimizing multi-robot systems.

3 Learning in Multi-Robots Systems

This chapter introduces how multi-robot systems can improve over time by learning efficient behaviours. It emphasizes three aspects of the learning scheme's diversity. First, how do robots have their goals defined, and how will it impact the resulting behaviours and learning ability. Second, what is the impact of the multi-robot architecture (decentralized, centralized, networked) and the information available for learning. Finally, how the learning algorithms use the goals and the architectures to improve the robot's behaviours.

3.1 Formal learning setting

Mobile robots are usually made of micro-controllers or complex computer systems that sense their environment with sensors and interact with it through actuators (Siegwart et al., 2011). The hardware is usually fixed and cannot be changed after deployment. What can be changed is the software that controls it. This software periodically reads sensor information that we will be called *observation*, and then must control the actuators by computing *action* parameters. Robots can therefore be considered as *agents* (Russell, 2010), and form the sense-act loop with the environment. Robots may also be able to communicate with other robots by sending messages. Figure 3.1 represents the different interactions between agents and their environment.

Most multi-agent systems where agents interact with an environment can be put in the formal setting of *Partially Observable Stochastic Games* or *Partially Observable*

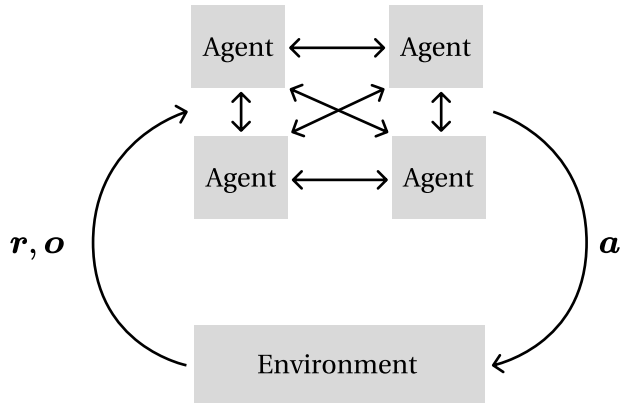


Figure 3.1: Interaction loop between the agents, their environment and teammates. The N agents each receive an observation $\boldsymbol{o} = (o^{(1)}, \ldots, o^{(N)})$ from the environment. When a task and a goal is defined, especially in the context of Markov Decision Processes, they can also receive a reward scalar $\boldsymbol{r} = (r^{(1)}, \ldots, r^{(N)})$. According to this information, the agents choose an action $\boldsymbol{a} = (a^{(1)}, \ldots, a^{(N)})$ to act on the environment.

Markov Games (Littman, 1994; Shapley, 1953a).

At each iteration t, the environment is described by a *Markovian* state s_t. It means that the state s_t encodes all the information necessary to determine the next state s_{t+1} of the environment. But the agents do not have access to this complete state. Instead, they receive an observation signal o_t corresponding to their input sensors' reading. This observation can very well represent the entire state of the environment or, on the contrary, be very sparse. According to this observation, they independently choose an action a_t to execute for the next step. Once all the actions are chosen, the environment model will use the current state and all agents' actions to compute the next state according to a transition function. And so on.

Formally, Partially Observable Stochastic Games (POSG) are defined as follows:

Definition 1 (Partially Observable Stochastic Games)**.** A Partially Observable Stochastic Game is a tuple $(N, S, \boldsymbol{O}, \boldsymbol{A}, \boldsymbol{R}, P)$, where:

- N is the set of agents indexed $1, \ldots, n$,
- S is the set of states of the game,
- $\boldsymbol{O} = \{O^{(i)}\}_{i=1,\ldots,n}$ are the sets of observations for each agents i,
- $\boldsymbol{A} = \{A^{(i)}\}_{i=1,\ldots,n}$ are the sets of actions for each agents i,
- $\boldsymbol{r} = \{r^{(i)} : S \times \boldsymbol{A} \to \mathbb{R}\}_{i=1,\ldots,n}$ are the immediate reward functions,

- $T : S \times \boldsymbol{A} \times S \rightarrow [0,1]$ is the stochastic transition function such that $\forall s \in S$, $\forall \boldsymbol{a} \in \boldsymbol{A}$, $\sum_{s' \in S} T(s, \boldsymbol{a}, s') = 1$

The transition function T gives the probability that all agents' actions $\boldsymbol{a}$ in state s at time step t will lead to state s' at step $t+1$:

$$P(s_{t+1} = s' | \boldsymbol{a}_t = \boldsymbol{a}, s_t = s) = T(s, \boldsymbol{a}, s') \tag{3.1}$$

In this manuscript, spaces, functions, vectors corresponding to several agents are noted in bold symbol. The superscript $^{(i)}$ specifies the agent i. The superscript $^{(-i)}$ is used to specify all agents except i.

Figure 3.2 represents a graphical model of a POSG between two states s_{t-1} and s_t. It highlights the relations between the different random variables of state, observation and action of an agent i and teammates $-i$.

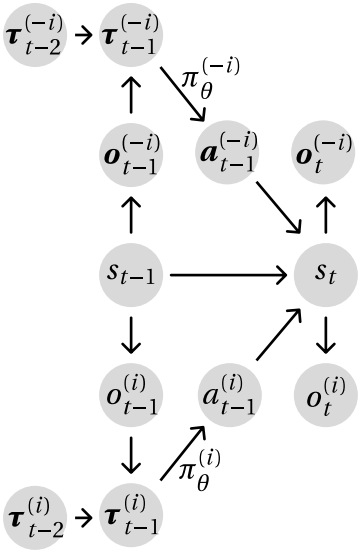


Figure 3.2: Graphical description of a Partially Observable Stochastic Game or Decentralized Partially Observable Markov Decision Process. Agents i gets an observation from the state, and append it on their action-observation history. This AOH is used by the policies to compute each agent actions. These action will then condition the next state, and so on. $(-i)$ indexation represents all other agents except i. Figure adapted from Q. Zhang et al., 2021.

POSGs are the most general models for agents interacting with an environment. However, taking inspiration from single agents Markov Decision Process (MDP) or

Partially Observable Markov Decision Process (POMDP) (Sutton and Barto, 2018), many models have been proposed over the years using some simplifying assumptions. For example, Multi-agent MDP extends MDP by assuming that all agents have the complete state observation and action space is jointly shared among all (Boutilier, 1996; Shoham and Leyton-Brown, 2008a). On the other hand, interactive POMDP (I-POMDP) (Gmytrasiewicz and Doshi, 2004) extend state space with behavioural models of others, and agents maintain beliefs over their models. In addition, many more models have been proposed depending on agents' knowledge of the system state and ability to communicate (Beynier et al., 2013).

Among them, a very popular model is the Decentralized Partially Observable Markov Decision Process (Dec-POMDP), which simplifies POSG by using a single reward function shared by all agents. Unfortunately, just like POSG, they are provably intractable (NEXP-complete, Bernstein et al., 2002), making them difficult to use when the number of agents is expected to be large.

Table 3.1, reproduced from Rizk et al., 2019, details some of the models by ranking their requirements in terms of scalability, heterogeneity and communication. Scalability is the ability of the model to manage a large number of agents without being computationally intractable. Heterogeneity is the capacity of the model to manage different agents. Furthermore, communication corresponds to the degree of communication required among agents.

Model	Scalability	Heterogeneity	Communication
Swarm Intelligence	High	Low	Low
Multi-agent MDP	Medium	Medium	Medium
Decentralized POMDP	Medium	Medium	High
Interactive POMDP	Low	Medium	High
POSG	Low	High	Medium

Table 3.1: Comparison of different Markov based decision-making models

Agents and policies The agents interact with the environment and, at each time step t, receive some observation $o_t^{(i)} \in O^{(i)}$ and choose some action $a_t^{(i)} \in A^{(i)}$. The list of these actions and observations along a trajectory can be compiled in an *action-observation* history (AOH) $\tau_t^{(i)} \in T = (O^{(i)} \times A^{(i)})^*$.

To choose an action depending on this history, agents use a *deterministic* or *stochastic policy distribution* $\pi^{(i)}$.

- A deterministic policy function returns, at each iteration t, the action $a_t^{(i)}$ based on the observation $o_t^{(i)}$ or action-observation history $\tau_t^{(i)}$;
- A stochastic policy distribution is a distribution on possible actions $a_t^{(i)}$ conditioned on $o_t^{(i)}$ or $\tau_t^{(i)}$.

These policy functions can take many forms. Recently, machine learning has revolutionized the optimization of single-agent systems, using *deep neural networks* as policy functions. Neural networks are functions composed of many interconnected artificial neurons, or nodes, which are organized into layers. They are used as function approximators and include many parameters or weights that can be adjusted during a learning process. Hence, we note $\theta^{(i)}$ the parameters of the neural network used by the policy $\pi_\theta^{(i)}$ of the agent i.

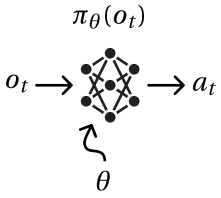


Figure 3.3: Deterministic policy represented as a neural networks. It outputs an action vector a_t depending on its observation input o_t and current parameters θ.

Evaluation We call any sampling or aggregation of sampling of the instantaneous reward function $r_t^{(i)}$ using the policy $\pi_\theta^{(i)}$, an *evaluation* $F(\pi_\theta^{(i)})$ of the policy of agent i. For example, such an evaluation can be the cumulative sum of instantaneous rewards during an episode, often called *return* function. To take into account the stochastic aspect of the environment and the agents' policies, the evaluation is, in reality, based on an estimate of the *expected return*, $\mathbb{E}_{\pi^{(i)}}[\sum_{t=0}^{T-1} r_t^{(i)}]$, where $r_t^{(i)}$ is the reward received by agent i at time step t, and T is the duration of the evaluation. Other types of returns can be used, like the *infinite-horizon discounted return* that discard rewards to tackle infinite horizon T, or less usually the maximum reward of an episode (Gottipati et al., 2020).

In multi-agent systems, the *optimal policy* of an agent depends on the policy of the other agents, thus introducing a game-theoretical aspect. We call the *best response*

policy the best policy of agent i by fixing the policy of the other agents $-i$. If none of the agents can change their policy without lowering its quality (as evaluated by the evaluation function), then the policies are in *Nash equilibrium* (i.e. all agents mutually play best response policies).

3.1.1 Conceptual challenges

The framework of POSG is very general and provides little information on the agents' constraints and the environment. However, the agents' sensori-motor capacities and the environment fundamentally impact the behaviour we expect from a single agent and the whole team. For example, during interactions, the spaces of observations and actions can restrain or favour the emergence of cooperation between agents. Moreover, during the optimization procedures, the reward function and its structure will fundamentally impact the policy obtained and the equilibrium reached by the team.

Local and global information

The amount of information shared between the agents, how it is shared and when it is shared will distinguish the possible learning algorithms and cooperation outcomes.

As we have seen, the task that the agents are brought to solve is globally described in POSGs by a state s_t. But the agents only have access to an observation $o_t^{(i)} = O^{(i)}(s_t)$ of it. This observation function will greatly impact the task and cooperative learning because it conditions the agents' perception.

In particular, the agents can sometimes observe global data that can be used as a reference and condition the coordination. For example, a synchronization signal can lead groups of agents as a conductor leads his musicians, or the traffic lights coordinate the departure of vehicles in intersections. These observations allow for indirect coordination without interactions.

Global or team-scale information can also be collected through message exchange. Communication allows sharing of information between agents. Depending on the bandwidth and network density, teams of agents can compute global information through distributed data measurement.

Credit assignment

Among this information is the reward of each agent, which has a very special place in the optimization process since it will be used as or by the loss function of the optimization. Its structure will then have a significant impact on the equilibrium reached and the behaviours that can be expected from the optimized agents. The question of credit assignment between agents in a team is tackled on Section 3.2.

Apart from the difficulty of sharing a specific reward, the problem of credit assignment can also come from the task description itself. We speak of a *hard exploration problem* when the rewards are very sparse (Andrychowicz et al., 2017), rare (Ecoffet et al., 2022) or deceptive (Lehman and Stanley, 2011). More precisely, a sparse reward signal is one that is infrequently encountered by the agent. This can make it more difficult for the agent to learn effectively because it receives less feedback about its actions. Deceptive reward, on the other hand, misleads the agent or leads it to make suboptimal decisions. For example, in a labyrinth task, if the reward is the distance from the agent to the exit, this might encourage the agent to go directly in the direction of the exit without taking the necessary detours to solve the labyrinth. Standard optimization methods usually perform poorly in such scenarios (Amodei et al., 2016; Bellemare et al., 2016), since random exploration is rarely efficient to discover successful states and obtain meaningful feedback.

Equilibrium and social optimum

Depending on the agent evaluation function characteristics, different equilibrium can be reached (Roughgarden, 2010). However, being in an equilibrium does not necessarily mean being in the best configuration for the team overall. Consider a prisoner's dilemma, Nash equilibrium situations do not give the best payoff from a utilitarian point of view. It risk to converge to mediocre stable states.

We wish to reach the *social optimum*, which is the point where *social welfare* (for example, the sum of individual assessments) is maximum. This point may not be an equilibrium, in which case, it would be non-stable.

We can use metrics to characterize games in terms of their equilibria and optima (Roughgarden, 2010). The price of anarchy is the ratio between the worst equilibrium

and the optimal centralized solution.

$$\text{PoA} = \frac{\text{Worst outcome of an equilibria}}{\text{Optimal outcome}} \tag{3.2}$$

If PoA is close to 1, all equilibria are good approximations of an optimal outcome and selfish behaviour are benign in these games. But a game with multiple equilibria may have a high price of anarchy even if only one of its equilibria is highly inefficient.

On the other hand *the price of stability* is the ratio between the best outcome of an equilibria and of the optimal outcome: (Schulz and Stier Moses, 2002)

$$\text{PoS} = \frac{\text{Best outcome of an equilibria}}{\text{Optimal outcome}} \tag{3.3}$$

A higher price of stability indicates a greater cost or effort required to maintain a stable equilibrium, and therefore a less stable game.

Other game theory concepts have been used over the years to describe and analyze strategy and equilibrium dynamics (i.e. whether and how equilibrium can be reached). Among these concepts is evolutionary game theory (Gintis et al., 2000). It aims at understanding how populations evolve over time in response to various strategies and environmental conditions. In evolutionary game theory, strategies are often modelled as biological traits that can be inherited from one generation to the next, and the success of a particular strategy is determined by its ability to reproduce and survive in a given environment. Evolutionary game theory refines the static Nash equilibrium concept with the notion of *evolutionarily stable strategies* (ESS). A strategy is an ESS if it is immune to invasion by mutant strategies, given that the mutants initially occupy only a tiny fraction of the population. These concepts have been extensively used to analyze the learning dynamic of multi-agent systems (Bloembergen et al., 2015).

Summary

When confronted with a new task and environment, it is essential to analyze these characteristics:

- How does the local sensing reflect the global state?
- Which information is shared between subsets of agents?
- How are agents rewarded during learning?

- And what equilibrium can be reached, depending on these characteristics?

Answering these questions allows for highlighting the differences in the agents' tasks and is critical for performance comparisons between the optimization methods that solve them.

3.2 The problem of rewarding agents

The concept of providing a purpose to artificial agents is closely tied to the definition of the reward signal. In standard reinforcement learning, fulfilling the purpose of the agent should coincide with maximizing this reward signal. This is known as the *reward hypothesis*:

> "[...] all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward)" (Sutton and Barto, 2018)

We generally speak of $r_t^{(i)} = r^{(i)}(s_t, \boldsymbol{a}_t)$ as the instantaneous reward function of agent i at time t, the function of the POSG that returns a real value for each state-actions pair. This function can be aggregated on a trajectory to form the *return function*:

$$R^{(i)}(\tau) = \sum_{t=0}^{T} r_t^{(i)} \tag{3.4}$$

where T is the finite horizon of the trajectory. In case of an infinite trajectory horizon, the concept of discounting is introduced to ensure the convergence of the infinite series. A $\gamma \in [0,1)$ is used for *discounted return* and value the rewards that are closer in time more than the farther ones:

$$R^{(i)}(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t^{(i)} \tag{3.5}$$

From these functions, one can want to compute the expected return of agent i following agents policies $\boldsymbol{\pi}$, $F(\boldsymbol{\pi}) = \mathop{\mathbb{E}}_{\tau\sim\boldsymbol{\pi}}[R(\tau)]$. This expectation is central to the optimization process and is often used as a *loss* or *fitness* function to be maximized.

One can also define the *on-policy value function*, $V^{\pi}(s)$, which gives the expected

return if the system start in state s and all agents act according to their policies $\boldsymbol{\pi}$:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau \sim \boldsymbol{\pi}}[R(\tau)|s_0 = s] \tag{3.6}$$

And the *on-policy action-value function*, $Q^{\pi}(s,a)$, which gives the expected return if the system start in state s, agents take an arbitrary action $\boldsymbol{a}$, and then forever after act according to policies $\boldsymbol{\pi}$:

$$Q^{\pi}(s,\boldsymbol{a}) = \mathop{\mathbb{E}}_{\tau \sim \boldsymbol{\pi}}[R(\tau)|s_0 = s, \boldsymbol{a}_0 = \boldsymbol{a}] \tag{3.7}$$

In all of these functions, it is their structures and dependencies on other agents that define which game is played, and the behaviours we can expect from agents (Agogino and Tumer, 2008)

3.2.1 Reward structure

We distinguish several types of games according to this reward function. On each side of the spectrum we distinguish purely competitive games from purely cooperative games. In the first case, the sum of the agents' rewards is zero, which implies that if an agent wishes to increase its own reward, it necessarily does so at the expense of the others. On the contrary, in purely cooperative games, each agent receives the same reward, which therefore represents the team's performance. Increasing its own reward, for an agent, means increasing the team's reward. Between these two extremes, we find general sum games. In these situations, interactions between agents can result in cooperative or competitive behaviour.

Figure 3.4: Spectrum collaboration of the reward function that condition the type of game that is played by the agents.

Pure competitive interests

Purely competitive games are usually modelled as zero-sum games. For the instantaneous reward, this translates into the following equation:

$$\sum_i r^{(i)}(s_t, a_t^{(1)}, \ldots, a_t^{(i)}, \ldots, a_t^{(n)}) = 0 \tag{3.8}$$

This means that if an agent manages to increase its individual reward, it is necessarily at the expense of others. In particular, when there are only two agents, the reward for one is the opposite of its opponent.

This framework has the advantage of feeding itself with complexity. When opponents find new strategies, it is necessary to find a new method to become better. Thus, even when starting with poor quality policies, agents will incrementally improve and exploit the weaknesses of their opponents, who will, in turn, correct these weaknesses to perform better, and so on. It is called exogenous auto-curricula in the terminology of Leibo et al., 2019.

Many of the most renowned achievements of multi-agent learning research have focused on pure-conflict games such as backgammon (Tesauro, 1994), chess (Campbell et al., 2002), go (Silver et al., 2016), two-player poker (N. Brown and Sandholm, 2019; Moravčík et al., 2017), StarCraft (Vinyals et al., 2019) and Stratego (Perolat et al., 2022).

Pure cooperative interest

In purely cooperative games, all agents receive the same reward:

$$\forall i, \quad r^{(i)}(s_t, a_t^{(1)}, \ldots, a_t^{(i)}, \ldots, a_t^{(n)}) = r(s_t, a_t^{(1)}, \ldots, a_t^{(i)}, \ldots, a_t^{(n)}) \tag{3.9}$$

One can place in this category, games where a team of agents must complete a given task and each agent receives a reward describing the team's performance, rather than their individual performance.

In contrast to the competitive case, cooperation tends to have a deleterious effect on learning. Indeed, finding a good answer to the policy of the other team members

will tend to discourage them from exploring. The team will then tend to over-fit and generalize poorly.

Mixed motivations

Between these two extremes, no restrictions are imposed on the reward functions and their inter-agent relationship, and the reward is said to be *mixed*. This *general-sum* case represents the vast majority of games in the realm of 2×2 matrix games (Robinson and Goforth, 2005). It is also the most general situation when agents are autonomous and goal-oriented. Each agent's rewards may conflict or be correlated and benefit from collaboration. Therefore, it is particularly in this context that the concepts of Nash equilibrium or evolutionary stable strategy (ESS) are important to characterize the interactions between agents.

In robotics, and particularly in the context of swarm robotics, agents are individually rewarded according to locally measured characteristics. Each agent typically has a self-assessment mechanism that estimates its performance. But this may induce local competition even is the reward scheme is designed with global welfare in mind (Bredeche et al., 2018).

Parallels can be found between the reward signal and resources in an environment. We can thus speak of excludable rewards to designate rewards withdrawn from the agent pool when one of them retrieves it. For example, in a situation where agents are tasked with gathering objects and are rewarded for each object they collect, global welfare is achieved when the agents distribute evenly across the objects. However, there may be local competition between individual agents, even if their overall goals are aligned.

3.2.2 Credit assignment and marginal contribution

Global reward does not scale well to difficult problems because the learners do not have sufficient feedback tailored to their specific actions. The same is true when the number of agent increases. It is hard to distinguish whose action allowed to get a reward. In cases where the reward signals are mixed or global, the designer wants them to promote social welfare while allowing learning for each individual. A compromise must therefore be found between these two problems:

- A global reward signal that promotes social welfare is not necessarily a good learning signal, especially when there are many agents. The marginal contribution must be extracted from it. In addition, the action of agents may require very close coupling to receive a reward.
- An individual reward signal will encourage competitive behaviour and may not result in a social equilibrium.

For example, let's imagine a one-team soccer task, i.e., a team of agents must coordinate to push a ball into a goal using only local sensor information. The team is rewarded with 1 if it scores, 0 otherwise. But this signal is very poor in information for agent learning. Especially in the extreme case where only one agent of the team is responsible for the performance, the learning algorithm will tend to reward and reinforce the other agents independently of their participation. The team risks falling into a local learning minimum and the overall performance will be difficult to increase. On the contrary, if we reward only the agent who scores the goal, we potentially avoid opportunistic apathetic behaviour, but the agents are now competing to score the goal and avoid that others score.

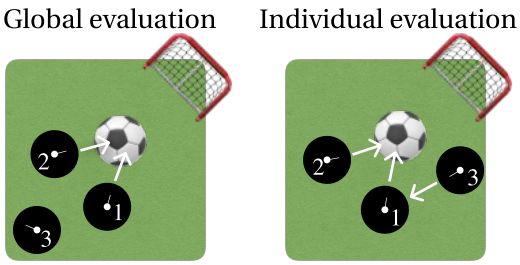


Figure 3.5: In this example, the three agents must push a ball into a goal to get a reward. On the left, a global reward rewards the team if the ball reaches its destination. This may allow the apathetic agent 3 to be rewarded without contributing to the task. On the right, an individual reward rewards each agent if it has contributed to pushing the ball. In this situation, competitive behaviour appears if an agent decides to prevent another agent from moving the ball to take credit for the goal.

It is therefore essential to find methods or methodologies to exploit all the information at our disposal to reward the agents of a team as faithfully as possible and to avoid the pitfalls of local minima. These methods can modify the reward signal r_t, marginalize the action-value function or directly alter the expected return F. The following paragraphs highlight some of the attempts to specify an individual evaluation in multi-agent systems, independently of their respective optimization algorithms.

Marginalization

The global idea of marginalization is to integrate out or remove a variable from a function. In other words, it is a way of taking a function of multiple variables and expressing it as a function of one or more of those variables, with the other variables marginalized or integrated out. This idea can be applied to a global evaluation that would depend on all team agents. The first approach is then to marginalize-out the effect of other agents on the evaluation to extract the so-called marginal contribution.

The *Difference Reward* (Agogino and Tumer, 2004a) is a translation in term of reward shaping (Ng et al., 1999) of the *Wonderful Life Utility* (D. Wolpert et al., 2000). The goal is to extract the true contribution of the agent by subtracting a counterfactual reward as if the agent performed a default action, to the gained reward.

$$d^{(i)}(s, \boldsymbol{a}) = r(s, \boldsymbol{a}) - r\left(s^{(-i)} \cup c_s^{(i)}, \boldsymbol{a}^{(-i)} \cup c_a^{(i)}\right) \tag{3.10}$$

Here s is the global state, $\boldsymbol{a}$ is the joint action of all agents and $r(s, \boldsymbol{a})$ is the global reward associated with them. $r\left(s^{(-i)} \cup c_s^{(i)}, \boldsymbol{a}^{(-i)} \cup c_a^{(i)}\right)$ is the global reward when agent i uses a default action $c^{(i)}$ instead of the action returned by its policy, and $s^{(-i)} \cup c_s^{(i)}$ is the state without i. $d^{(i)}$ is then the advantage of using the policy instead of a default action for the agent. This allows extracting marginal contributions that perform better for learning as it is more sensitive to the individual agent's action. It is also aligned to the global system performance, as the increase of difference reward by changing an action $a^{(-i)}$ on a given state s, is equal to the increase of global reward:

$$\frac{\partial d^{(i)}}{\partial a^{(i)}}(s, \boldsymbol{a}) = \frac{\partial r}{\partial a^{(i)}}(s, \boldsymbol{a}) \tag{3.11}$$

But it requires two evaluations of the global reward function, which is not usually possible in practice. Colby et al., 2016 propose to learn an approximation $\tilde{r}$ of the global reward to evaluate the second part of the equation:

$$\tilde{d}^{(i)}(s, \boldsymbol{a}) = r(s, \boldsymbol{a}) - \tilde{r}(c_s^{(i)}, c_a^{(i)}) \tag{3.12}$$

$\tilde{d}^{(i)}$ is then the difference between the global reward function and the approximated reward function if agent i took a default action.

Also, as the number of agents increases, the amount of information each must process increases, reducing the effectiveness of the difference rewards. To address this shortcoming, it is possible to use a hierarchical organization to reduce the amount of information exchanged between agents (HolmesParker et al., 2016). Here, agents are divided into smaller teams and coordinate to optimize that sub-team's goal. Then, a control agent is assigned to each sub-team and must coordinate to optimize the global goal.

The effectiveness of this method has been demonstrated in various domains, including air traffic control (Cruciol et al., 2013), rover navigation (Agogino and Tumer, 2008; Knudson and Tumer, 2010), satellite coordination (Agogino et al., 2012a), distributed sensor networks (HolmesParker et al., 2013; Turner, 2006), communication system optimization (Agogino et al., 2012b) and for the coordination of robots swarm (Douchan et al., 2019).

Difference reward has, more recently, inspired methods for the evaluation of counterfactual action-value functions, such as in counterfactual multi-agent policy gradient (COMA) (Foerster, Farquhar, et al., 2017). It marginalizes agents' individual actions by applying a counterfactual baseline function that uses a centralized critic.

Sometimes the calculation of the marginal contribution is not sufficient to represent everyone's contributions. For example, consider a game where the team N gets an evaluation of $F(N) = 1$, and all the agents are required for this success. The marginal contribution of each agent i is therefore $F(N) - F(N \backslash \{i\}) = 1$, which does not represent the involvement of each agent in the task.

The Shapley value (Shapley, 1953b) allows to calculate this contribution in a fair way by averaging the counterfactual marginal contribution on all possible coalitions. If we denote F the evaluation function of a coalition, the Shapley value φ_F of agent i is given by:

$$\varphi_F(i) = \frac{1}{n} \sum_{S \subseteq N \backslash \{i\}} \begin{pmatrix} n-1 \\ |S| \end{pmatrix}^{-1} (F(S \cup \{i\}) - F(S)) \tag{3.13}$$

The sum is performed on all subset S of the whole team N not containing agent i.

For the above example, all sub-team smaller than the full coalition have a null evaluation ($\forall S \neq N = \{1, \ldots, n\}, F(S) = 0$, and $F(\{1, \ldots, i, \ldots, n\}) = 1$). The computation of the

Shapley value thus reduce to:

$$\varphi_F(i) = \frac{1}{n} \tag{3.14}$$

Each of the n agents receives an equal share of the evaluation that corresponds well to their contribution.

The Shapley value has the following properties:

- *efficiency*: the sum of the values of each agent is equal to the global payoff;
- *symmetry*: two substitutable agents have the same value;
- *linearity*: if the task can be divided in multiple tasks, the Shapley value of the whole is the Shapley value of the sum;
- *null agent*: a player that contributes nothing have a null Shapley value

Although more accurate, this contribution estimate is computationally expensive, especially as it grows factorially as the number of agents increases. Several methods have been proposed to approximate this calculation, giving up some properties. For example, L-Shapley and C-Shapley (Chen et al., 2018) only consider local interactions on a graph of agents, which breaks the efficiency properties. Others have used a priori knowledge to simplify the computation but break the axiom of symmetry (Frye et al., 2020; Heskes et al., 2020). The use of the Shapley value with computational approximation has been experimented by Li et al., 2021 on a StarCraft multi-agent reinforcement learning task.

Decomposition

Another approach is to decompose the global reward by a learned, task-specific, *value decomposition function*. We place ourselves in the setup where the system is evaluated by a joint action-value function $Q(\boldsymbol{\tau}, \boldsymbol{a})$ that must be decomposed into marginal action-value functions to allow the agents to learn. The simplest method is to assume that the joint action-value function is the sum of the marginal action-value function of each agent (Sunehag et al., 2018).

$$Q(\tau^{(1)}, \ldots, \tau^{(n)}, a^{(1)}, \ldots, a^{(n)}) = \sum_{i=1}^{n} Q^{(i)}(\tau^{(i)}, a^{(i)}) \tag{3.15}$$

It is also possible to use, for example, a neural network to map a joint value function to individual value functions as used in the QMIX methods (Rashid et al., 2020; Rashid

et al., 2018). The method frees itself from the additivity constraint but requires monotonicity on the relation between the joint action-value function and the individual action-value function so that the argmax of each agent is the global argmax.

$$\underset{\boldsymbol{a}}{\text{argmax}} \quad Q(\boldsymbol{\tau}, \boldsymbol{a}) = \begin{pmatrix} \text{argmax}_{a^{(1)}} Q^{(1)}\left(\tau^{(1)}, a^{(1)}\right) \\ \vdots \\ \text{argmax}_{a^{(n)}} Q^{(n)}\left(\tau^{(n)}, a^{(n)}\right) \end{pmatrix} \tag{3.16}$$

Furthermore, the QTRAN algorithm discards these assumptions of additivity and monotonicity in the factorization and allows any non-linear combination of value functions (Son et al., 2019).

However, these methods may suffer from a lack of interpretability, and the architecture of the decomposition function limits the possible decomposition.

Cooperation signalling

In some cases, it is necessary to have cooperative behaviours to obtain a reward. We can then use signals to indicate to the agents which behaviours will allow them to obtain a reward, even though they have not yet learned to cooperate together. d_{++} (Rahmattalabi et al., 2016) is another reward shaping strategy that provide incentives for agents to perform action whose reward may depend a lot on others' actions. We say in this case that the agent's actions require a high degree of coupling. For example, in a task where m agents may need to take simultaneously several photos of the same point of interest to gain a reward, d_{++} will reward one agent for taking one picture even if no other agent is present. If agent i adds n agents for the state-action pair $\boldsymbol{a}$, the $d_{++}^{(i),m}(\boldsymbol{a})$ reward is given by:

$$d_{++}^{(i),m}(\boldsymbol{a}) = \frac{r\left(\boldsymbol{a}_+\left(\cup_{i=1,\ldots,m}\right)\boldsymbol{i}\right) - r(\boldsymbol{a})}{m} \tag{3.17}$$

This will allow the agent to be rewarded even if all conditions are not fulfilled and provide *stepping stone* rewards to learn an effective behaviour faster than without this help.

In our point of interest photography case, if three agents are required to take a picture, only one agent is currently in a proper position, and taking a picture gives a reward of

1, the $d_{++}^{(i),2}$ reward that adds $m = 2$ agents will gives $1/2$. $d_{++}^{(i),m}$ is usually computed across multiple m values to get the most reward that requires adding the less virtual agents. Here $d_{++}^{(i),1} = 0, \quad d_{++}^{(i),2} = 1/2, \quad d_{++}^{(i),3} = 1/3, \ldots$, which justifies the choice of $m = 2$.

Peer feedback

Even in the case where the reward is not global, reward attribution can be imprecise. For example, suppose a reward is measured locally at time t. In that case, an agent may have prevented a teammate from obtaining a reward or, on the contrary, may have contributed to its obtaining without benefiting from it. Evaluating one's own contribution in a group can then be based on the evaluation given by teammates. This is called peer feedback. For example, in the case of networked agents, one can share an estimate of their impact on their neighbours' rewards. They can then evaluate the losses or gains they may have made on other agents and modify their rewards accordingly (Hostallero et al., 2020; Jaques et al., 2018).

3.3 Learning architectures

Now that we have described the different ways of receiving rewards, it remains to study how to obtain more efficient policies. As seen previously, multi-robot systems are very diverse. For example, the composition of teams, their communication and computational capabilities, and the presence of a hierarchy or a central authority will influence the type of architecture possible for policy learning. This section describes some of these architectures.

Generally speaking, the goal is to optimize a policy function, either deterministic $\pi^{(i)} : O^{(i)} \to A^{(i)}$, or stochastic $\pi^{(i)}(a^{(i)}|o^{(i)})$, for each agents. We need distinguish two stages:

- the *learning phase*, where policies are updated,
- the *execution phase*, where the agents fix and use their policies to interact with the environment and each others. It allows gathering data and reward for updating policies in the learning phase.

The two stages can take place at different times or intertwine with each other.

In this section, we explore the differences in the information structure of the optimization method (K. Zhang et al., 2019). In Chapter 2, we described the different interaction structures between agents, from decentralization to total centralization. These differences in structure also have an impact on policy learning. And notably, which entity has access to which information for the *learning phase* and *execution phase*? The answer to this question is crucial and will largely constrain the type of algorithm used for optimization and the structure of the policy.

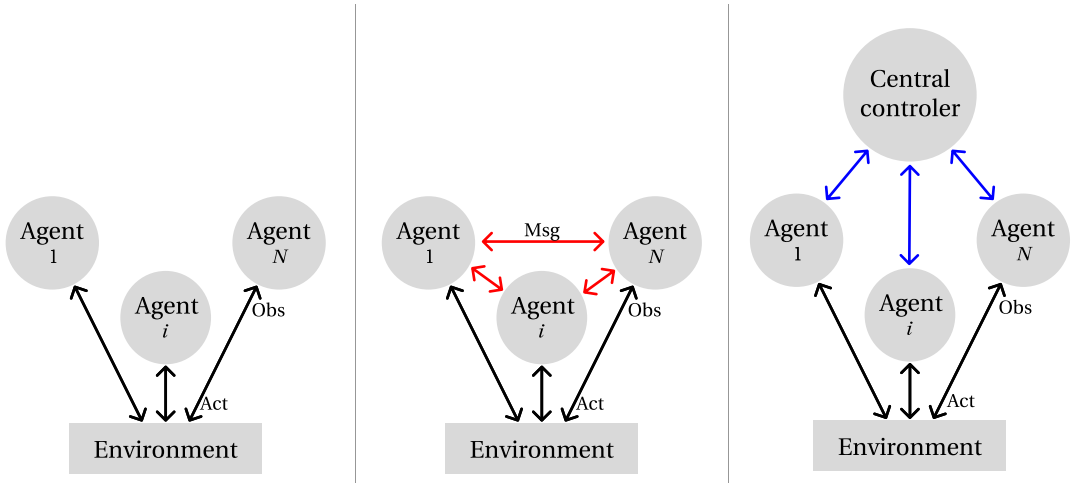
Tan, 1993 studied the differences between learning agents that share information during learning and ones that do not. From his experiments, he found three essential guidelines: (1) additional sensations from another agent are beneficial if they can be used efficiently, (2) sharing learned policies or episodes is beneficial at the cost of communication, and (3) for joint tasks, agents engaging in partnership can outperform independent agents.

A learning algorithm is centralized when information flows through a single central node. The most extreme case of centralization is when a central policy chooses a vector of joint actions to be distributed over all agents. The so-called *centralized training centralized execution* (CTCE) paradigm allows using standard single agents optimization techniques. However, the term multi-agent system is debatable in this case since the intelligence is concentrated in a single agent. In every other case, the agents are decentralized, at least during their interactions with others and the environment. They also may use communication channels to share information locally or reach a consensus. We distinguish three commonly used information flow structures for learning and executing multi-agent systems:

- Decentralized learning;
- Networked learning;
- Centralized learning, decentralized execution

3.3.1 Decentralized learning

In general, a central controller for execution and learning does not exist. Each agent is independent and embodied in its environment. Formally, each agent i as only access to its own observation $o^{(i)}$, actions $a^{(i)}$ and reward function $r^{(i)}$ during the learning phase and execution phase. Figure 3.6.a represents the iterations between the agents



(a) decentralized learning (b) networked learning (c) centralized learning, decentralized execution

Figure 3.6: (a) Decentralized agents interacting with an environment. (b) Agents interacting with an environment and using communication channel during learning and execution. (c) Agents interacting with an environment and using a central authority for learning

and the environment.

All agents still interact with each other through the environment. And obtaining a reward signal may also depend locally on the actions of other agents. A coupling of behaviours can therefore be achieved through the optimization process and cooperation can emerge in a goal-oriented fashion. The great advantage of this structure is that it does not suffer from the *curse of dimensionality* when the number of agents increases because each agent only manages its own observation and action.

Learning without explicit information sharing is often called *independent learning* (Claus and Boutilier, 1998; Tan, 1993). Each agent ignores the multi-agents aspect of the systems and learns an independent policy function. This leads to great instability because the environment appears non-stationary from the point of view of any agents as they independently update their policies, violating Markov assumptions required for convergence (Laurent et al., 2011).

When the task is stochastic, agents need to distinguish between different sources of variation in the observed rewards. It can be due to the noise in the environment or to the behaviours of the other agents (Matignon et al., 2012).

In particular, important variation can be caused by the exploration of other agents. In single agent reinforcement learning, exploration-exploitation trade-off requires to balance between the exploitation of the agent's current knowledge and the exploration of new actions to improve that knowledge. But in the multi-agent case, exploration of an agent can destabilize the learned policies of other independent learners (Matignon et al., 2010). This is the *alter-exploration problem*.

This also can leads to *shadowed equilibria*, where local observability and non-stationarity cause locally optimal actions to become globally sub-optimal (Fulda and Ventura, 2007; Panait and Tuyls, 2007).

3.3.2 Networked learning

To address the convergence problem of independent learners in decentralized systems, we can allow the agents to transmit information through communication channels. However, it is important to distinguish between several types of communication:

- The communication used by an optimization process and whose syntax and semantics are defined by the system designer. It is used to improve the performance of a policy by modifying its parameters.
- The communication used by agents to inform or motivate behaviour. Its syntax and semantics can be fixed or learned. It is used by the policy as an input.

Agent networks have been extensively studied in the context of agent consensus building. Consensus means reaching an agreement on specific interest quantities that depend on the state of all agents (Olfati-Saber et al., 2007). A consensus algorithm is an interaction rule specifying how to exchange information on a communication network. It can be considered as a form of learning since agents are forced to adapt some of their parameters to a situation. For example, we can think of consensus formation or social rules in human crowds (e.g. right or left avoidance to avoid a collision in Moussaïd et al., 2011). This convention building is thus a form of social learning.

For example, let's have a network of N agents that must reach a decentralized agreement, or consensus, on a particular issue. The agents can only communicate with their neighbours, as defined by the set $N^{(i)}$ for each agent i. To reach this consensus,

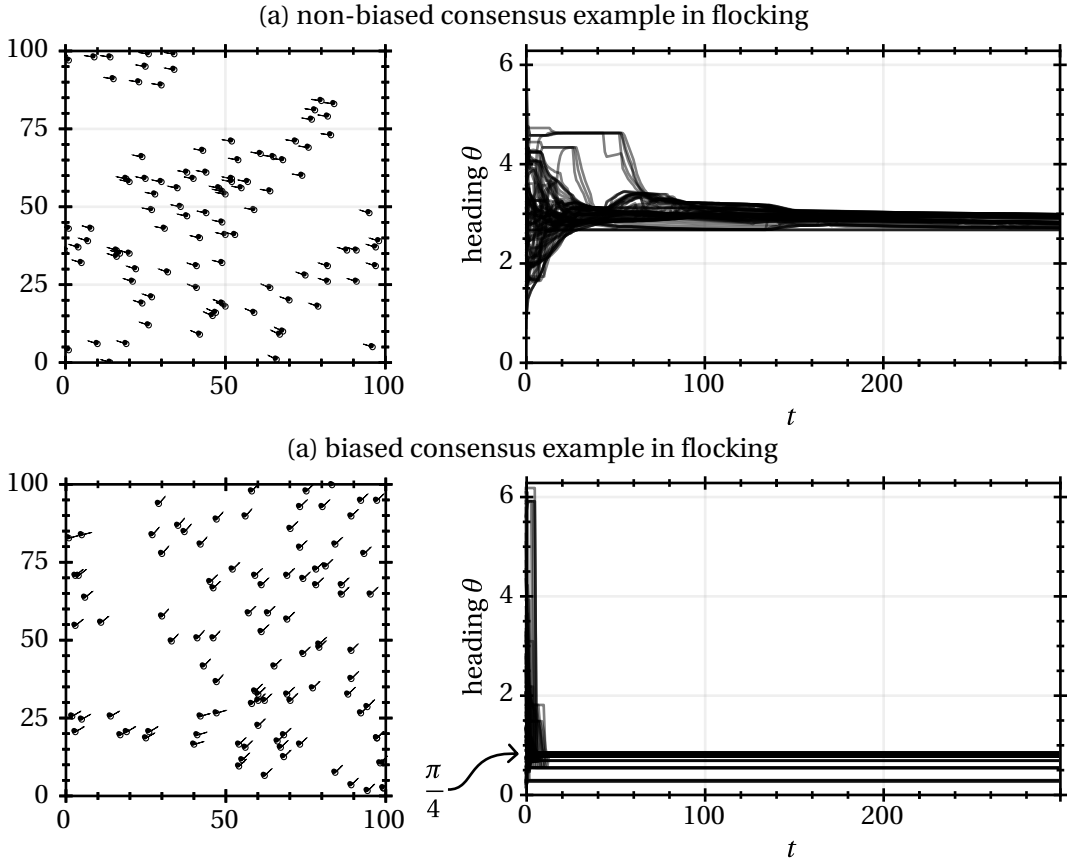


Figure 3.7: Results of two flocking experiments. In both case, agents are randomly initialized on a 100×100 arena with random orientation. They move at constant velocity. In (a), at each time steps they share their orientation with their neighbours and update theirs according to equation 3.18. In (b), each agent assess its orientation parameter θ with an objective function higher for value close to $\pi/4$. The agents then filter message of agents that have lower assessment than them.

the agents use a discrete-time update rule given by:

$$\theta_{t+1}^{(i)} = \frac{1}{1+|N^{(i)}|}(\theta_t^{(i)} + \sum_{j \in N^{(i)}} \theta_t^{(j)}) \tag{3.18}$$

where $\theta_t^{(i)}$ represents the value of the consensus variable for agent i at time t. Under this update rule, if the network topology is fixed, it can be shown that the values of $\theta_t^{(i)}$ will converge to the average of the initial values $\theta_0^{(i)}$ for all agents as t tends toward infinity. However, in many scenarios, such as flocking, the topology of the network may vary with time and state. In these cases, it may not be possible to guarantee convergence to a global average value. But still, each encounter will drive the team

toward an agreement. Figure 3.7.a represents the result of such scenario.

One way to introduce a global goal in such scenarios is to bias the agents' learning using a defined objective function. Each agent will only accept updates from its neighbours if it has a better evaluation according to the objective function. For example, if the objective is to have a heading close to $\pi/4$, then the agents will tend to align their heading to this value when they encounter a better neighbour. This will help bias the agents toward a common goal, even when the network topology changes. Figure 3.7.b represents the result of such scenario. This concept is the foundation of the embodied evolution algorithms (Bredeche et al., 2018) described in Section 3.5.1.

3.3.3 Centralized learning, decentralized execution

Even if agent policy execution is decentralized, learning can greatly benefit from additional information to converge faster towards the social optimum. In particular, many algorithms benefit from centralized policy learning.

Formally, during the execution phase, each agent i as only access to its own observation $o^{(i)}$, actions $a^{(i)}$ and reward function $r^{(i)}$. But during the learning phase, a central controller can aggregate the agents' data to compute and store additional information. This learning scheme comes from work done for planning on *Decentralized Partially Observable Markov Decision Process* (DEC-POMDP) or *Partially Observable Markov Games* (POMG) (Kraemer and Banerjee, 2016; Oliehoek et al., 2008), and has become very popular in recent work on Multi-Agent Reinforcement Learning (see Section 3.4.1).

This configuration is particularly suitable when the environment is simulated or when a central controller with high bandwidth communication channels is available. Figure 3.6.c represents the interactions between the agents, the environment, and a central controller. The controller can range from a simple reward assignment method to a full critics in the Actor-Critics framework (Foerster, Farquhar, et al., 2017; Perolat et al., 2018; Sunehag et al., 2018) and compute lots of centralized information.

One can also imagine cases where agents could communicate periodically with a centralized system via communication channels. This raises the question of how much information to transmit to the central controller and its relevance. The Cooperative

Co-Evolutionary Algorithms (CCEA) (Ma et al., 2019; Potter and De Jong, 1994a) are particularly suitable when not much information can be exchanged between agents. The degree of centralization is, in this case, represented by the importance of the interactions that the evolutionary algorithms will have between them to submit new parameters to be evaluated. For example, there is little centralization if the agents independently submit new parameters at each new evaluation. On the other hand, if the different agents coordinate, for example, to precisely evaluate each agent, the impact of centralization is more significant.

3.4 Policy optimization algorithms

On top of these different information flow architectures, multiple policy optimization methods can be used (Tuyls and Stone, 2017). The goal for each agent is to maximize the probability of rewarding trajectories. If we note F the expectation of gain associated with the parameters θ of the policy π_θ, we have:

$$F(\theta) = \int_\tau P(\tau \mid \pi_\theta) R(\tau) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} [R(\tau)] \tag{3.19}$$

Here, $P(\tau \mid \pi_\theta)$ is the probability of the trajectory τ following the policy π_θ, and $R(\tau)$ is the return or cumulated reward along the trajectory, as described in equation 3.4.

Then, the optimal policy is the one that maximize this function

$$\pi_\theta^* = \arg\max_\theta F(\theta) \tag{3.20}$$

There exists many algorithms to solve these problems. Recently, reinforcement learning methods and their multi-agent variant (MARL) have significantly gained popularity. But many other optimization methods based on evolutionary algorithms offer good performances for systems with limited computation and communication capabilities.

3.4.1 Multi-agent reinforcement learning

The reinforcement learning framework is the most commonly used for policy optimization in interaction with Markov processes (Sutton and Barto, 2018). RL algorithms are first differentiated by their use of a model of the environment. Models allow agents to

plan ahead and predict what could happen after their actions. But such information is usually not accessible or hard to learn, especially in the multi-agent setting (Q. Zhang et al., 2021). A field of research focuses on modelling opponents for example using fictitious play (G. W. Brown, 1951) or rational learning (Kalai and Lehrer, 1993). But in this manuscript, we will focus solely on model-free algorithms as they allow more simple yet effective algorithms. In the realm of model-free RL, two global approaches are used: *Q-learning* and *policy gradient*.

Q-learning

The goal of Q-learning is to iteratively find an approximation of the on-policy action-value function (Mnih et al., 2015; Watkins, 1989). The action selection is then performed by a greedy policy that chooses the action that maximizes the found action-value function for a given state.

Q-learning can be applied to the multi-agent setting but several problems arise, among which:

- it requires full-observability of the state and agent's actions, which is rarely the case in the multi-robot or swarm settings,
- the environment appears non-stationary from the view of any agents as they independently update their policies, violating Markov assumptions required for convergence (Laurent et al., 2011)

Tan, 1993 was the first to use Q-learning in a multi-agent configuration and introduce the Independent Q-learning (IQL). This framework considers all agents to be independent, and the other agent's actions are treated as part of the environment. Each learns its own action-value function that conditions only on its observation and its own action. This approach has been successfully applied to deep reinforcement learning on a two-player pong task (Tampuu et al., 2017). But it still suffers from the non-stability issue, and independent Q-learners may fail to distinguish between exploration by teammates and stochasticity in the environment (Claus and Boutilier, 1998).

In Tesauro, 2003, each agent's state space is augmented with an estimate of the other agents' policies allowing the Q-function to be made stationary. Foerster, Nardelli, et al., 2017 enable the use of an experience replay memory which is core for deep

Q-learning methods.

Policy gradient

Another class of reinforcement learning algorithms is policy gradient methods. At each iteration of the learning phase, the agent parameters take a step in the direction of the gradient loss function $\nabla_\theta F(\theta)$:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta F(\theta) \tag{3.21}$$

where α is the learning rate. Using the action-value function previously defined, this gradient can be written as follow (Sutton et al., 1999):

$$\nabla_\theta F(\theta) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}\left[\nabla_\theta \log \boldsymbol{\pi}_\theta(a \mid s) Q^\pi(s,a)\right] \tag{3.22}$$

Here, $Q^\pi(s,a)$ is unknown and needs to be approximated. This approximation is called a *critic* and leads to a variety of *actor-critic algorithms* (Sutton and Barto, 2018). Many researchers have used centralized learning and decentralized execution architecture to tackle this problem. For example, the *multi-agent deep deterministic policy gradient algorithm* (MADDPG) (Lowe et al., 2017) computes an individual critique for each agent using all agents' observations and actions. Since then, many methods have been proposed to compute this action-value function, including the use of decomposition function in VDN (Sunehag et al., 2018) and QMIX Rashid et al., 2020; Rashid et al., 2018 or counterfactual estimation (Foerster, Farquhar, et al., 2017).

But policy gradient methods are known to have high variance gradient estimates. This is especially true in the multi-agent case, where the probability of taking a gradient step in the right direction can decrease exponentially with the number of agents (Lowe et al., 2017).

3.4.2 Direct policy search

Instead of using value estimation and computing a costly gradient, many algorithms simply search for an optimal policy directly over its parameter space. This requires much less information transfer between the agent and a central authority and much

less granularity in the reward function (Kober et al., 2013). One of these, evolutionary algorithms, are general-purpose optimization algorithms loosely inspired by natural evolution (Eiben, Smith, et al., 2003). They have a long history of being used for multi-agent systems and robotic applications (Baldassarre et al., 2003; Nolfi and Floreano, 2000; Trianni, 2008).

Namely, evolutionary algorithms use the concepts of selection and variation to explore the complex space of candidate solutions to design a policy. The main idea is to create a set of candidate solutions and make them converge toward more efficient solutions. The set of solutions is called a *population*. Each of the candidates in the population will be evaluated on their ability to solve the desired task. This evaluation is called a fitness function and can take many forms. In the case of this thesis, it is generally the cumulative reward accumulated during an evaluation period T. Thus:

$$F(\theta^{(i)}) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta^{(i)}} \left[R^{(i)}(\tau) \right] \tag{3.23}$$

Based on this assessment, some candidates will be selected to become the next generation of parents. Parents will thus be used as a basis to generate a set of offspring through *mutation* and *crossover* operations. The crossover operator takes two or more candidates and produces a new one.

$$\theta^{(1,2)} = \textsc{CrossOver}(\theta^{(1)}, \theta^{(2)})$$

The mutation operator takes one candidate and produces a new one.

$$\theta^{(1)'} = \textsc{Mutation}(\theta^{(1)})$$

These two functions can take many forms depending on the structure of θ and the specificity of the algorithms in use.

The set of offspring is, in turn, evaluated, and the next generation is then selected from the union of all parents and offspring. It is the survivor selection. Finally, the survivors are used as a base population for the following algorithm loop. Figure 3.8 summarizes the whole evolutionary algorithm procedure.

This procedure will thus optimize a population of candidates, allowing to keep diversity in the solutions envisaged and therefore gain in robustness. When tackling

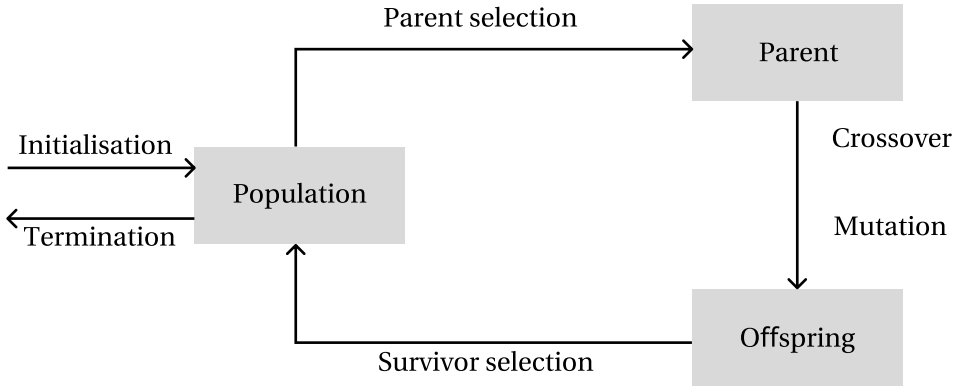


Figure 3.8: Principle of an evolutionary algorithm

multiple agents, the use of an evolutionary algorithm asks the question of the team composition (homogeneous or heterogeneous) and level of selection (individual or team-wise) (Waibel et al., 2009). The optimal combination depends on the amount of cooperation required by the task.

3.5 Evolutionary algorithms

This generic procedure of evolutionary algorithms can be adapted to many configurations. In this thesis, we will use in particular two very different classes of evolutionary algorithms, embodied evolution (EE) and cooperative co-evolutionary algorithm (CCEA). This section briefly describes how they work.

3.5.1 Embodied evolution

Embodied evolution (Bredeche et al., 2018; Watson et al., 2002) (EE) is a framework of online networked learning method for designing collective behaviours in swarm-like collectives using an evolutionary algorithm scheme. Each robot has an embedded optimizer that must optimize a policy to maximize a reward function that is locally defined in each robot. The principle of embodied evolution is recalled in Figure 3.9.

At initialization, each individual is initialized with its own random parameter set, called *active parameters*. They are then used by the policy to determine the actions to be taken by the robotic agent. Thanks to this policy, the robot can have a sense-act

Figure 3.9: In embodied evolution, agent are on-line learning neural network parameters by exchanging message through a communication network managing them with an evolutionary algorithm.

loop commonly used in robotics. As it moves through its environment, it evaluates its quality through the reward function and, therefore, how well its behaviour is aligned with the task to be performed. When two robots meet, they exchange their active parameters and their assessment of their policies. This operation is called *mating*. The choice of a partner may be purely based on environmental contingencies, but other considerations may play a role, such as performance or similarity. Each robot then stores the shared information in a parameter reservoir for future use. The robots will then execute an evolutionary algorithm on all the parameters harvested during their life cycle. Thus, a new set of parameters is created or selected based on all the information locally collected by the robot. Then the reservoir is cleared for future collection.

3.5.2 Cooperative Co-Evolutionary Algorithms

On the other side of the spectrum, teams of robots can also be optimized via a centralized evolutionary algorithm. Cooperative Co-Evolutionary Algorithms, first introduced by Potter and De Jong, 1994a, are popular evolutionary algorithms for optimizing multi-agent systems. In contrast to traditional evolutionary algorithms, which evolve a single population of solutions, cooperative co-evolutionary algorithms evolve multiple populations of solutions simultaneously. These populations are called subpopulations, and are evolved cooperatively, in the sense that the fitness of each solution in a subpopulation depends on the solutions in the other subpopulations.

The main initial idea of these methods is to decompose an original problem into a set of lower dimensional and tractable sub-problems that can be solved individually using an evolutionary algorithm. For example, we may want to maximize a fitness function F that depends on parameters $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} F(\boldsymbol{\theta}) \tag{3.24}$$

We can decompose $\boldsymbol{\theta}$ into a set of $(\theta^{(1)}, \ldots, \theta^{(n)})$. Then the problem is fully separable, if we have:

$$\underset{\theta^{(1)},\ldots,\theta^{(n)}}{\operatorname{argmax}} F(\boldsymbol{\theta}) = \left(\underset{\theta^{(1)}}{\operatorname{argmax}} F\left(\theta^{(1)}, \ldots\right), \ldots, \underset{\theta^{(n)}}{\operatorname{argmax}} F\left(\ldots, \theta^{(n)}\right) \right) \tag{3.25}$$

We can apply this methods to a multi-agent problem, where each agent i deals with its own policy parameters $\theta^{(i)}$ and the team must maximize a fitness function $F(\theta^{(1)}, \ldots, \theta^{(n)})$.

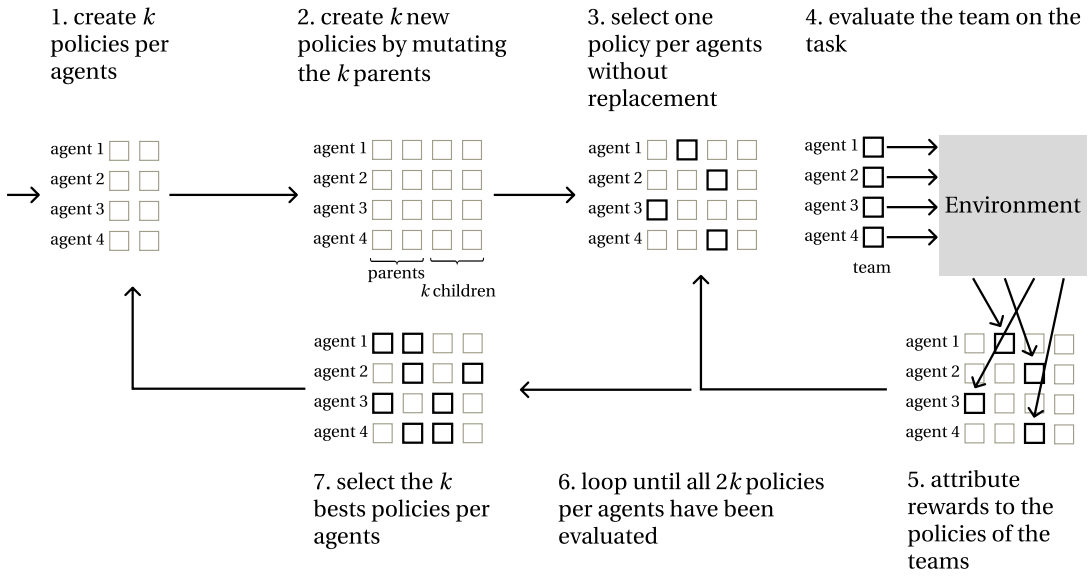


Figure 3.10: Diagram of a typical CCEA algorithms.

Figure 3.10 describes the general functioning of a CCEA algorithm. For each agent, the optimization is done through an independent evolutionary algorithm. They all maintain k version of their parameters $\theta^{(i)}$. From these k parents, they derive k new $\theta^{(i)}$ called children. Each agent has, therefore $2k$ version of their parameters. Then, all

agents submit one version and receive an evaluation of the version. This loops until all $2k$ are evaluated at least once. Depending on these evaluations, they each select the k parameters that performed best. The whole process then loops until a stopping criterion is met.

3.6 Thesis objectives

As we have seen in previous sections, the diversity of MRS induces many agent goal definitions, information flow between agents and, thus, optimization algorithms. Figure 3.11 distinguishes two axes to represent the information flow:

- between the agents and a centralized authority (x-axis);
- between each other through a communication network (y-axis).

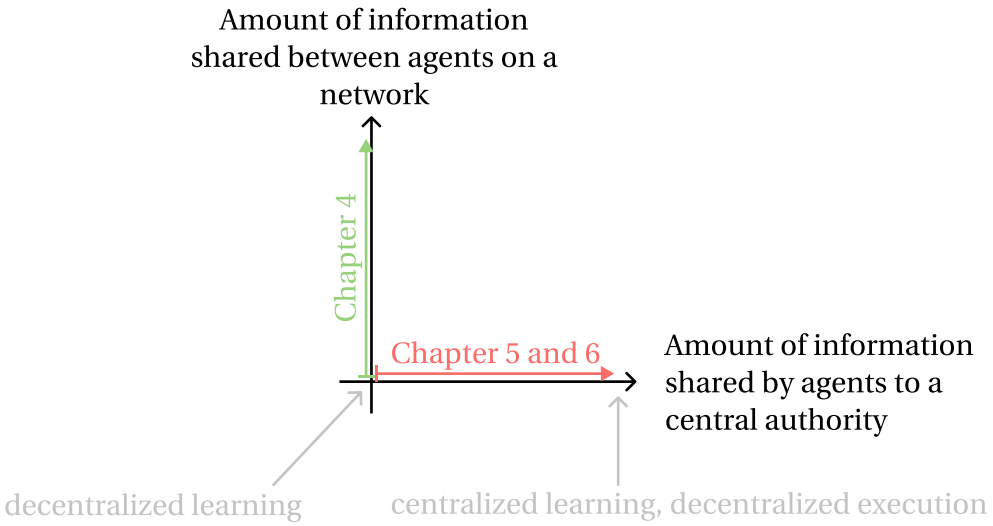


Figure 3.11: Representation of the amount of information shared between agents and a central authority.The decentralized learning architecture and the centralized learning architecture, decentralized execution as described above, are marked in grey in this diagram for localization purpose.

The flow of information is not uncorrelated to the agent goal definition. For example, a centralized system is easily associated with a team-wise global evaluation in the cooperative case. On the contrary, when the system is decentralized or networked, the evaluation is usually measured locally, leading to mixed or competitive evaluations.

The objective of this thesis is to explore the individual and group learning dynamics

in evolutionary collective robotics. We explore two distinct collective robotics setups regarding the reward system, information flow and learning algorithm. Their common denominators are to involve an evolutionary optimization loop and modulate the amount of information shared between agents during the learning process.

In Chapter 4 we propose a novel evolutionary algorithm from the class of *Embodied Evolution* (Bredeche et al., 2018; Watson et al., 2002) for a networked team of agents that must learn from a local evaluation during their lifetime. As represented in Figure 3.11, the algorithm can be used on various network densities and modulate of the amount of information transferred between agents.

In Chapters 5 and 6, we will focus on the class of *Cooperative Co-Evolutionnary Algorithm* (CCEA) that use a centralization for episodic learning agents that receive a global team-wise evaluation. As represented in Figure 3.11, the algorithm we propose will modulate the amount of information and learning synchronization between agents and a centralized authority.

4 Networked Agent Lifetime Evolution in Swarm Robotics with Limited Communication Bandwidth

This chapter is adapted from:

- Fontbonne, Nicolas, Olivier Dauchot, and Nicolas Bredeche. "Distributed online learning in swarm robotics with limited communication bandwidth." *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2020.
- Bredeche, Nicolas, and Nicolas Fontbonne. "Social learning in swarm robotics." *Philosophical Transactions of the Royal Society B 377.1843 (2022)*: 20200309.

4.1 Introduction

Swarm robotics is a sub-domain of collective robotics. This field is characterized by the use of a generally large number of robots with limited communication and computation capabilities, and an objective defined at the level of the whole swarm (e.g. exploring the environment, searching for resources, collectively transporting heavy objects). The challenge is to design the rules of microscopic interactions between the robots in order to achieve a relevant swarm organization at the macroscopic level (Bayindir, 2016; Beni, 2005; Brambilla, Ferrante, et al., 2012; Hamann, 2018).

There are several bio-inspired methods to address this problem of distributed programming, which can be grouped into two main classes: manual programming and automatic design methods. In the first case, the aim is to explicitly reproduce at a more or less abstract level behaviours observed in nature, but whose macroscopic

result is sometimes difficult to foresee in advance.

In the second case, the use of evolutionary optimization algorithms makes it possible to automate the design of individual behaviours according to a global objective function (Trianni et al., 2008). Indeed, since the structure of the environment is not known in advance, writing the objective function makes it possible to specify the general objective of the task (e.g. maximizing the number of objects picked up for a foraging task), without giving any precision on the structure of the strategy to be deployed. This is a well-known problem framework in evolutionary robotics: the structures of the research space and the fitness landscape have a tenuous relationship, for which stochastic optimization methods are a good fit.

However, an important assumption that underlies manual or automatic design methods is that behavioural rules are obtained in the laboratory, and then deployed in a real-world situation as is without any further tuning. In other words, the environment and the nature of the task are considered stationary between the conditions known at the time of design and the conditions actually encountered afterwards.

In this chapter, we are interested in a different class of problem. We consider that the environment in which the robots will be deployed is not known in advance. Provided a general objective, that can be written down as an objective function defined at the level of the individual (e.g. given a foraging task, each robot should try to get the largest number of items), we implement a networked on-line learning algorithm to allow the swarm of robots to progressively acquire the necessary skills to perform the target task. The behaviours learned depend on the nature of the task, but also on the particularities of the environment. For example, picking up objects in a foraging task requires different strategies depending on whether the objects are grouped together in a specific area or distributed in the environment.

The class of evolutionary algorithms that addresses this problem is known as either embodied evolution or social learning. Whether biological or cultural evolution is considered, both methods can be seen as instances of designing algorithms inspired by evolutionary dynamics for swarm robotics. These methods stand as on-line networked learning that takes into account interactions between hardware limited robots distributed over a possibly large space with local communication. On the one hand, these evolutionary dynamics methods are similar to the classical evolutionary robotics approach as the goal is to optimize a black-box objective function whose analytical

form is not known. On the other hand, they differ in that the problem of the transition to reality is simply non-existent in evolutionary dynamics methods: the actual learning starts at the time of operational deployment of the robots, and not before it. In other words, the goal is to design robust on-line learning algorithms as much as robust solutions.

To date, several evolutionary dynamics algorithms have been validated on real robots, addressing various scientific and practical issues. However, the number of robots is often in the order of ten(s) due to the hardware and CPU requirements of such robots to implement learning algorithms (e.g. 20 e-Puck robots in Bredeche et al., 2012, 6 Thymio-2 robot with a Raspberry board in Heinerman et al., 2016). This is in stark contrast with other works in swarm robotics that do not implement learning capabilities, where the number robots is often counted by the hundreds (Slavkov et al., 2018; Valentini, Ferrante, Hamann, et al., 2016a)- and even up to slightly more than one thousand robots (Michael Rubenstein and Nagpal, 2014). The challenge remains open as to implementing on-line networked learning on such a large swam of robots.

In this chapter, we propose a new networked on-line learning algorithm inspired by evolutionary dynamics. The originality of this algorithm is to minimize the requirements in terms of memory and communication cost, in order to be deployed on a swarm of very low cost robots with limited computing and communication capabilities (e.g. a Kilobot robot). The proposed algorithm is based on the *horizontal gene transfer* mechanism observed in bacteria: when two robots interact, a part of the control parameters is transferred to the robot's memory and communication system. Thus, the amount of information transferred can take into account time and bandwidth limitations, regardless of the number of control parameters governing decision making.

Though similar ideas have been explored in evolutionary computation (Harvey, 2009), it has never been employed in the context of networked on-line learning, where the possibility of modulating the amount of information transmitted between robots makes it possible to take into account hardware and environmental constraints. Obviously, the price to pay for such an algorithm is a reduced convergence speed. But it also makes it possible to adjust the amount of information exchanged to account from practical limitations at hand. Indeed, the quality of the communication bandwidth depends both on technical characteristics and environmental contingencies (number

of packet collisions that increase with the number of robots, disturbances due to materials used in the environment, etc.).

The rest of the chapter is organized as follow : Section 2 presents the algorithm and evolutionary operators. Section 3 describes the experimental setup. Section 4 presents results on a classic foraging task, first by comparing the proposed algorithm with a state-of-the-art counterpart, and second, by performing a sensitivity analysis of the various hyper-parameters of the algorithm. We then present an extension of the algorithm to automatically adapt to the available communication bandwidth.

4.2 Algorithm

Both embodied evolution (Bredeche et al., 2018; Watson et al., 2002) and social learning (Heinerman et al., 2015) implement an evolutionary algorithm scheme, adapted to perform networked on-line learning as illustrated in Figure 4.1. Each robot optimizes a policy to maximize a fitness, F, that is locally defined in each robot.

Figure 4.1: Principle of embodied evolution algorithms. The robotic agent sends and receives parameters of a policy function, and uses a evolutionary algorithm to improve it own active parameters.

Robots follow a *sense-act loop* commonly used in robotics for reactive agents. Each robot i runs a deterministic policy $\boldsymbol{\pi}_{\boldsymbol{\theta}_i}$. At initialization, a robot is initialized with a random control parameter set $\boldsymbol{\theta}_i$, referred to as *active parameters*. These parameters are then used by the policy to determine the actions to be taken by the agent in reaction to its observations.

The typical algorithm goes as follow : as a robot moves through its environment, the

quality of its behaviour is assessed using a *fitness* function which depends on the user-defined task to be achieved (e.g. number of items gathered for a foraging task, which is also sometimes referred to as *fitness value*, *reward*, *performance* or *utility*). When two robots are within communication range, they exchange their active parameters and the (current) assessment of their policies. This operation is called *mating*. The choice of accepting a partner may be purely based on the environmental contingencies but other consideration may play a role such as performance or similarity. Mating does not automatically imply a change in the active parameters of one agent, but incoming data is generally stored for later use in a *reservoir*.

The renewal of a robot's active parameters typically occurs after some predefined amount of time. At this point, the robot will use information stored in the reservoir to update its current active parameter set, which imply constructing a new candidate set of parameters from the reservoir using typical selection and variation evolutionary operators (Bredeche and Montanier, 2010; Fernandez Pérez et al., 2014; Hart et al., 2015; Heinerman et al., 2015; Prieto et al., 2009).

4.2.1 Horizontal Information Transfer

All embodied evolutionary algorithm to date assume that the whole set of active parameters (and current fitness) can be sent as a single communication packet. This is generally true where the bandwidth is virtually infinite (as in simulation) or where it is order of magnitude larger than the size of messages to be sent (as with Linux-running robots using WIFI exchanging a few hundreds neural network weights) (Bredeche et al., 2018). Using hardware-limited robots such as Kilobots (Rubenstein et al., 2012) then raises the question of limiting the number of control parameters to stay within the limits of the communication bandwidth, which can be due to either technical limitations or environmental contingencies. For example, Kilobots not only use slow IR communication (a few octets per seconds), but also are limited when the number of Kilobots close-by increases due packet collisions.

In order to decouple communication and computation constraints, we introduce the HIT algorithm. HIT stands for Horizontal Information Transfer, and can be seen as an instance of either embodied evolution or social learning algorithms. It uses evolutionary operators in a networked on-line fashion, and manages communication between robots by exchanging a part, rather than all, of the robot's control parameters.

By exchanging only partial information, we expect two benefits:

- the possible recombination of behavioural skills (i.e. efficient subsets of the policy parameters) obtained by separate robots, which would not be possible in a winner-take-all approach. This will be shown is Section 4.4.2;
- the decoupling of the number of control parameters that can be *used* for control and *sent* to other robots. In other words, it makes it possible to exploit the computation and memory capabilities of the robot, without being limited by its communication capability. This will be shown is Section 4.4.3.

In addition, HIT differs from other similar algorithms as it does not require a reservoir to store incoming information. Upon receiving control parameters and current fitness from a nearby robot, a robot will immediately integrate the new parameter values (i.e. overwriting the current corresponding values) if its interlocutor's fitness is better.

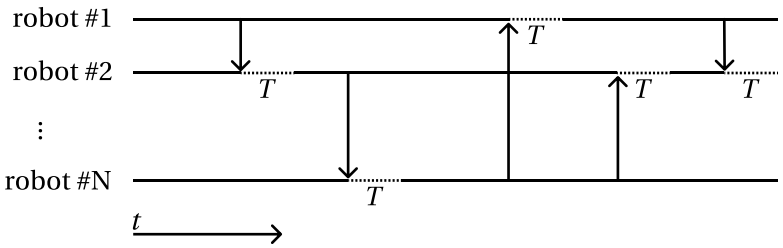


Figure 4.2: Communication or mating between the agents. Each robot has a maturation period of time T where communication with neighbours is turned off. A new maturation period is initiated after each modification of the active parameters. During this period, the robot only evaluates its policy

Algorithm 1 describes HIT as implemented in each robot. It includes both evolutionary learning and decision-making. It can be decomposed in two cycles:

- *lines 4-7, sense-act cycle*: The agent retrieves information from the environment via its sensors using the sense() function. It retrieves an observation vector o and a reward scalar r. The reward is stored in a queue, here $Rf[.]$, of size T for later use. As for the observation vector, it is used by the deterministic policy $\pi_{\boldsymbol{\theta}}$ to compute the action vector $\boldsymbol{a}$. Finally, the act() function is responsible for transmitting the commands to the actuators.
- *lines 9-15, evolutionary cycle*: After an evaluation period of fixed size T, the agent can enter the evolutionary cycle. It broadcasts a random subset of its control parameter set to its neighbours with an evaluation of its quality (i.e. the fitness).

The subset size is controlled by the transfer rate α. When it receives a new message (indicated by the $new\_message$ variable), if the received parameters have a better fitness, then it replaces its own control parameters by the sender's selected parameters (function TRANSFER). Then it applies a Gaussian mutation of variance σ on all its parameter set (function MUTATION).

These two cycles are also clearly represented of Figure 4.3.

Algorithm 1: The HIT algorithm (*Horizontal Information Transfer*)

Data:
α : transfer rate $\in [0,1]$,
T : evaluation time,
π : Policy function,
$\boldsymbol{\theta}$: Random uniform initialisation of policy parameters, $\dim(\boldsymbol{\theta}) = m$,
$R[T]$: Empty reward buffer of size T,
r : Null reward scalar,
$\boldsymbol{a}$: Null action vector,
o : Null observation vector

```
1  begin
2      t = 0
3      while True do
4          o, r = sense()
5          R[t mod T] = r
6          a = π(o|θ)
7          act(a)
8          if t > T then
9              F = Σ_{k=0}^{T-1} R[k]
10             Create the Idx array by drawing randomly αm integers in range [0, m − 1]
                without replacement
11             broadcast(θ[Idx], Idx, F)
12             if new_message then
13                 θ = TRANSFER(θ, F, Idx_message, θ_message, F_message)
14                 θ = MUTATION(θ)
15                 t = 0
16             end
17         end
18         t = t + 1
19     end
20 end
```

In the following, we provide more details on the selection and variation operators.

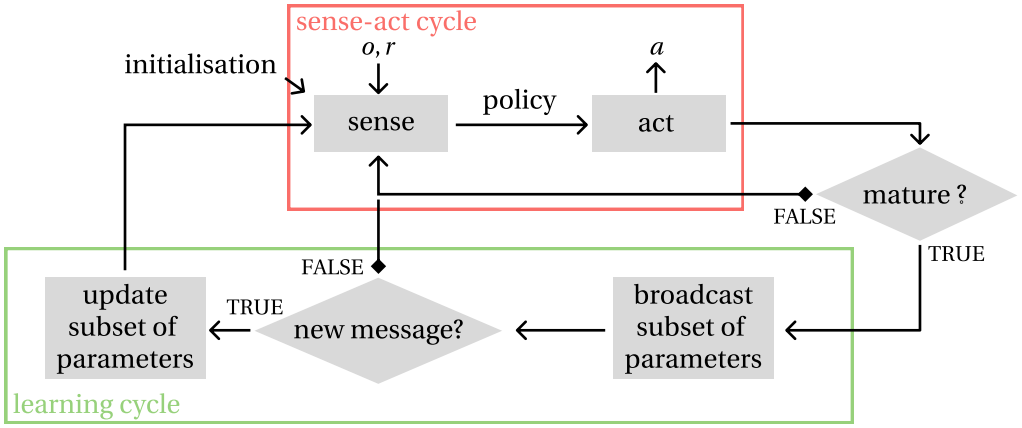


Figure 4.3: Diagram of the HIT algorithm. The red box represents the sense-act cycle, where the robot interacts with the environment with actions a to receive observations o and rewards r. The green box represents the learning cycle which consists in sending and receiving subsets of parameters when the robot is mature. The robots being deployed in the environment, the two cycles take place simultaneously.

4.2.2 Evaluation and Selection

The assessment of individuals depends mainly on the quality of the policy but can be very noisy due to non-stationary stochastic variabilities in the environment and behaviour of other agents. Depending on the definition of the fitness function and the distribution of the rewards in the environment, some individuals may find themselves naturally favoured by chance and thus spread misguided policies.

HIT uses a sliding time window of size T to assess the robot's performance. The window size depends on the task and the environment at hand, and must be set so that sufficient information is gathered to provide a relevant estimate of the quality of current policy. In order to compare policies in a fair manner, robots can only exchange information after the sliding window has been completely filled. We call this *maturation period*, which duration corresponds to the time required for a full self-evaluation, i.e. the evaluation time T.

Figure 4.2 illustrates the dynamics of the algorithm. Whenever two robots meet, the worst-scoring robot replace part of its control parameter with those of the best-scoring robot. Then, the updated robot resets its fitness and enters a maturation period during which communication is disabled.

At each step, the agent receives a reward R_t . We call fitness F_t, the cumulated reward obtained by the agent during T steps (sliding window). The evaluation time T, the fitness F_t and reward R_t at step t are linked by the relation:

$$F_t = \sum_{k=0}^{T-1} R_{t-k} \tag{4.1}$$

While a longer evaluation time should allow for more accurate assessment of a given policy's quality, there is of course a cost in terms of convergence speed.

4.2.3 Transfer operator

HIT introduces a *transfer operator* with rate $\alpha \in [0,1]$ that defines the amount of information that will be transferred during communication between two robots. As an example, a transfer rate of $\alpha = 0.5$ means that half of the control parameters will be randomly selected to be sent. From one interaction to another, a different subset of parameters can be selected for sending, and two interacting robots will send the same quantity of possibly different parameters.

There can be different methods to randomly pick the parameters to be sent. A basic method, which we use afterwards, is to send n parameters stored in 32-bit float, along with an additional n bytes to send the indexes of these parameters. However, several strategies can be used to compress or limit the quantity of information without compromising overall optimization. It can also possible to send a segment of parameters whose offset changes randomly in-between each new message.

Algorithm 2 details the transfer mechanism during the mating operation. Unlike other embodied evolution algorithms, HIT does not store incoming information in a reservoir. Whenever the fitness of the sender is greater or equals to the fitness of the receiver, the received parameter are directly used to overwrite the corresponding local parameter.

Algorithm 2: *Transfer function*

Data:

$\boldsymbol{\theta}$: active parameters,

G: current evaluation,

$\text{Idx}_{\text{message}}$: received parameter indexes,

$\boldsymbol{\theta}_{\text{message}}$: received parameters,

F_{message}: received evaluation

```
1 begin
2     if F_message > G then
3         for i ∈ Idx_message do
4             θ[i] = θ_message[i]
5         end
6     end
7 end
```

4.2.4 Mutation operator

HIT implements a classic *Gaussian mutation* operator. It applies a perturbation centred on the current parameter value, with a variance σ. It is defined as follow, for the m parameters:

$$\theta[i] \leftarrow \mathcal{N}(\theta[i], \sigma) \quad \forall i \in [1, m]$$

Mutation allows to introduce and maintain some level of diversity during the exploration of the parameters space. While it may not be useful in the first steps of evolution, it eventually maintains some level of diversity for exploring the parameters space as using the transfer operator can only leverage what is already present in the initial population.

4.3 Experimental setup

4.3.1 Task and Environment

In order to study the dynamics of HIT, we devise a foraging task, similar to tasks solved by many species of insect collectives. It is also a good abstraction of a search and

retrieve robotic task, where an unknown environment must be explored to retrieve specific objects or resources.

The goal here is for each robot to collect as many items as possible. Both robots and items are initially randomly placed in the arena. Whenever a robot picks up an object at iteration t, it gets a reward of $r_t = 1$, and a new item appears at a random location in the arena to maintain a constant number of resources.

4.3.2 Simulation environment

We used the Roborobo3 simulator (Bredeche et al., 2013), which is a pseudo-realistic, light and fast multi-agent simulation environment developed in C++. It provides a pseudo-realistic physics robotic model similar to the seminal Khepera2 and e-Puck 2-wheeled mobile robots while still ensuring fast enough simulation to allow for extensive experimental work involving hundreds of robots.

Robots, objects and the environment are physically represented by bitmap images, which allows roborobo3 to manage collisions at the pixel level, though location, perception and displacement are handled in the continuous domain.

Robots have a size of 5px × 5px . They move in an environment of 1400px × 800px that is uniformly filled with objects.

4.3.3 Robot Model

Robots are subject to a kinematic model, as described in Figure 4.5. It is, therefore, a question of controlling a velocity vector. Thus, the robots have a 2-dimensional action space A where the two dimensions represent speed ($a_0 \in [-1,1]$ for [backward, forward]) and angular speed ($a_1 \in [-1,1]$ for [clockwise, anti-clockwise]).

Their observation space O is composed of 16 range sensors that get information about the surrounding of the agent. They are ray-casting sensors that have a maximum range of three times the robot length (15px). For each of these sensors, four information are extracted: the distance to contact if an object is detected, and three other Boolean information for each sensor to explicit the type of information (object, wall or agent). Thus, we obtain an observation vector of $\dim(O) = 64$ dimensions.

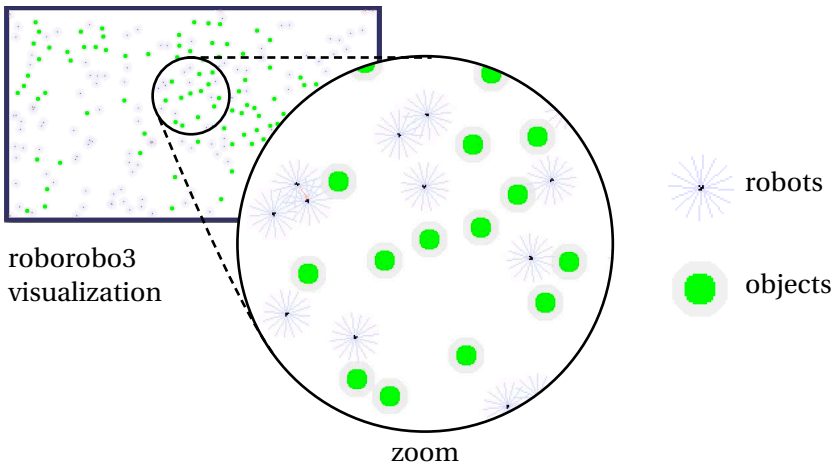


Figure 4.4: Arena used for the experiments. It features 150 robots (small blue dots) with 16 short-range sensors and 100 items (green dots). Items disappear when caught, to reappear at a new random location. Robots are never relocated, and the HIT algorithm runs as a networked on-line fashion.

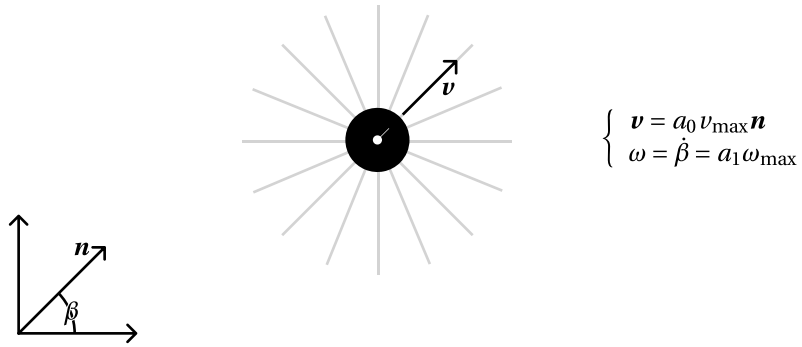


Figure 4.5: Robot model with 16 sensors, a velocity $\boldsymbol{v}$, a maximal speed $v_{\max}$, an angular speed ω and a maximal angular speed $\omega_{\max}$. The control variables are then a_0 and a_1.

The robot's policy maps observations $o \in O$ to actions $\boldsymbol{a} \in A$. For that purpose, we use a multi-layered Perceptron (MLP) as the main policy structure. Figure 4.6 details the full topology between inputs and outputs, and the number of parameters.

This architecture imposes a large number of free control parameters that need to be optimized. In the present case, this means 1074 parameters.

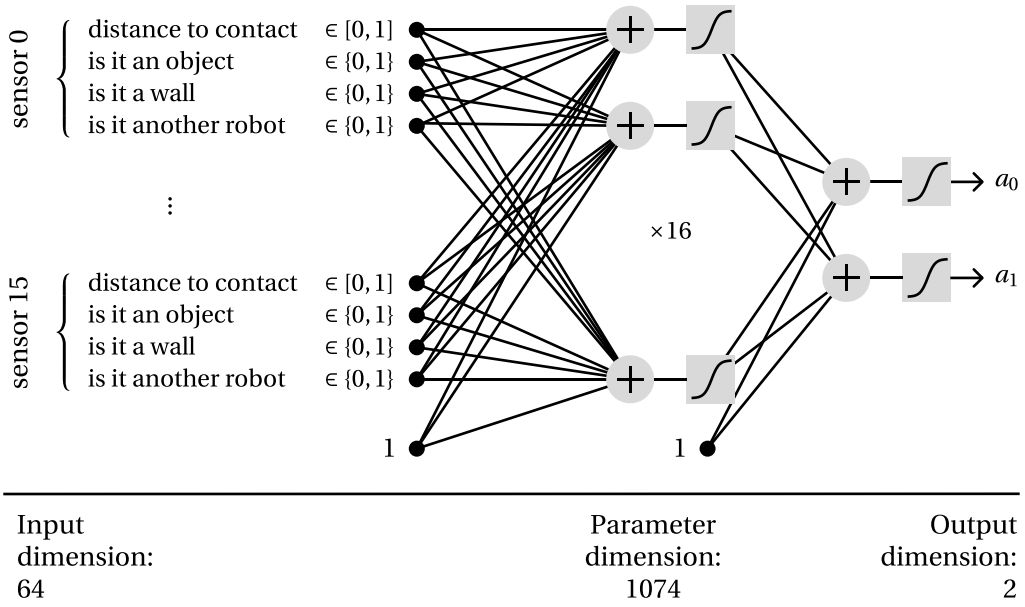


Figure 4.6: Architecture of the policy function. Each sensor gives information about the distance to obstacle and the type of object detected, if any. The architecture represented here is a multi-layered Perceptron, used for the experiments. It has 64 dimensions as input, a hidden layer of 16 dimensions plus a bias term for each layer. The action dimension is 2. The total number of parameters is 1074.

4.4 Result

In this section, we conduct an experimental study of the HIT algorithm. Values for all experimental parameters can be found in Table 4.1, including environmental, neural network controller and HIT parameters. The rectangular arena used is represented in Figure 4.4, with 150 robots and 100 items.

4.4.1 Qualitative and Quantitative Evaluation

We analyse the dynamics of HIT while solving the foraging task for a well chosen set of meta-parameters (See next section for an extensive analysis of the α and σ meta-parameters). To analyse the results, we define two notions:

- the *final fitness*: to assert the quality of a particular algorithm, we measure the fitness after convergence;
- the *characteristic time*: to evaluate the speed of convergence, we measure the

Parameter	Value
Environment parameters	
Population size	150
Number of objects	100
Arena size	1400px × 800px
Robot size	5px × 5px
Sensor length	15px
Maximum velocity $v_{\max}$	2 px/steps
Maximum angular velocity $\omega_{\max}$	30 degrees/steps
Controller: multi-layered Perceptron	
Initialisation range	$[-400, 400]$
Sensory inputs	64
Hidden layer	1
Hidden size	16
Control outputs	2
Total number of parameters	1074
Controller: simple Perceptron (only Sec. 4.4.2)	
Initialisation range	$[-400, 400]$
Sensory inputs	163
Control outputs	2
Total number of parameters	328
HIT parameters	
Evaluation time T	400
Transfer rate α	varying
Mutation size σ	varying

Table 4.1: Parameters

time at which the average fitness is half the final fitness. This is approximately the position of the inflexion point of the data sequence that plots the median fitness. This was chosen *a posteriori* as in all the experiments we conducted, we *always* observed a sigmoid-like increase of the fitness when switching from the initial low fitness values to the final fitness values.

Figure 4.7 compiles the results obtained with 128 replicates of the HIT algorithm, with $\alpha = 0.8, \sigma = 0.001, T = 400$. Starting with low values, the fitness increases between 40000 and 100000 steps (characteristic time ~70000), and then converges to a stable value (final fitness of ~2.4).

It should also be noted that the variance after convergence is rather small, advocating for the robustness of the algorithm. This is actually confirmed by comparing HIT

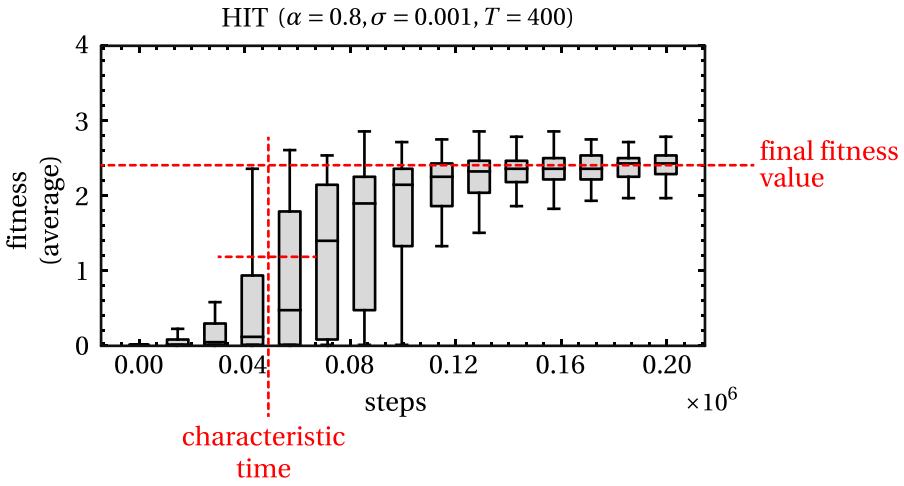


Figure 4.7: Results with HIT, $\alpha = 0.8, \sigma = 0.001, T = 400$. The parameters used for these simulation are described on Table 4.1. The Y-axis represents the distribution of the average fitness among all 150 agents, over all runs. Results are compiled from 128 independent simulation runs.

with a canonical state-of-the-art embodied evolution algorithm, which we refer to as VanillaEE (Fernandez Pérez et al., 2014; Hart et al., 2015; Montanier et al., 2016).

Figure 4.8 shows the results in term of characteristic times and final fitness of two variants of the HIT algorithm (HIT($\alpha = 0.3$), which is expected converge slower, and HIT($\alpha = 0.8$), as shown before) and of the best-shot of VanillaEE we could find (i.e. $\sigma = 0.001$ mutation rate, elitist selection). To account for the implementation difference between HIT and VanillaEE[I], we re-evaluate the final fitness by extracting the control parameters from the last generation, and then running these with the learning algorithm deactivated. All claims are backed using Mann-Whitney U Test.

Firstly, the final fitness is roughly similar for both HIT variants, which is expected. Both display also an advantage over the VanillaEE control algorithm. Secondly, the characteristic time shows, as expected, that HIT($\alpha = 0.3$) provides the slowest convergence speed. HIT($\alpha = 0.8$) and VanillaEE converge faster, which is actually true *on average*. However, VanillaEE displays a higher variance both in terms of convergence speed and final fitness when compared to HIT($\alpha = 0.8$).

Actual behaviours for a typical run are shown in Figure 4.9. At the very beginning,

[I]The original implementation of VanillaEE is synchronous, meaning that all robots update their policy at the same time, which HIT does not do.

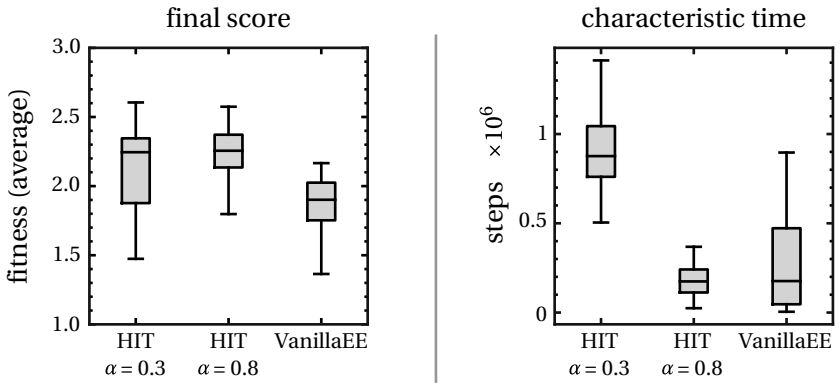


Figure 4.8: Comparison between HIT and VanillaEE all with $\sigma = 10^{-3}$. The box plots are computed with 80 independent runs. The characteristic time measures the step at which the average fitness reaches half its final value. The final fitness is the average fitness after saturation.

robots fail to either meet with one another or to capture objects other than by chance (Figure 4.9-(a)). At the end of learning, robots wander around, covering large areas and following non linear trajectories (Figure 4.9-(b)). Trajectories from typical behavioural strategies are illustrated in Figures 4.9-(c-d). Each figure displays the trajectory of one robot and shows its interaction with other robots and objects (e.g. the robot turns toward a detected object). These two figures illustrate the result of both the implicit exogenous selection pressure (robots meet with one another as it favours diffusion of behavioural strategies) and the explicit endogenous selection pressure (robots are drawn to items as it increases their total reward).

This change in behaviour is also captured by looking at the evolution of the amount of communication during the course of evolution. For a typical run of HIT(α = 0.8), Figures 4.11 and 4.10 respectively show the evolution of fitness and number of messages exchanged through time. Comparing the two figures reveals that the increase in communication between robots actually precedes the increase of fitness values, and remains stable throughout the experiment, even before the final fitness value is reached.

4.4.2 Tradeoff between Speed and Accuracy

Both the transfer rate α and the mutation size σ can have an impact on learning speed and quality. In this section, we provide an extensive analysis of these two

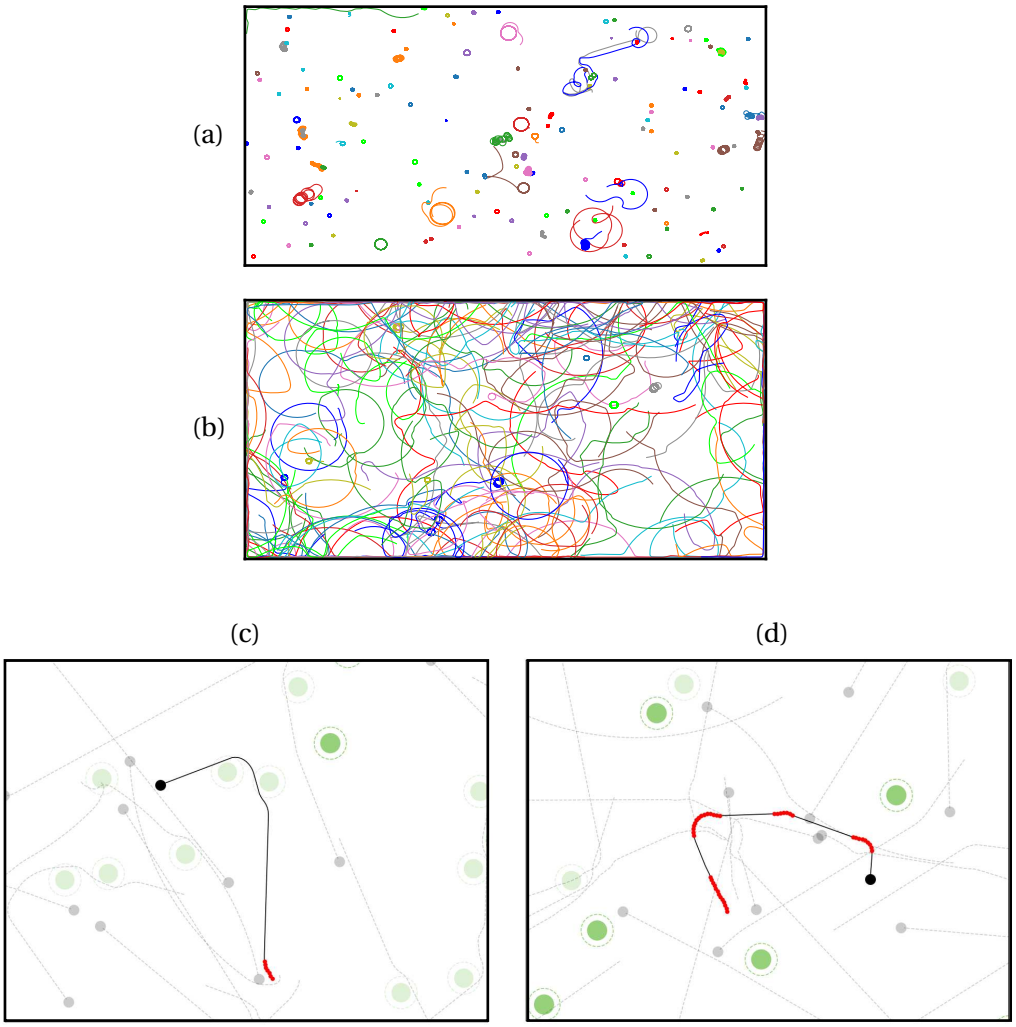


Figure 4.9: **(a)**: initial trajectories of 150 robots in the first time steps of learning (0−400 time steps, items are not displayed). **(b)**: trajectories of 150 robots at the end of the simulation (13600 − 14000 time steps). **(c)** and **(d)**): two examples of robot trajectories produced after learning, for an arbitrary selected focal robot (small black circle) and its trajectory during the last 100 time steps (black or red line - red denotes time steps where the focal robot shared information with a nearby robot). The figures also show other robots (small grey dots) and items (large green dots - lighter green denotes an item that has been captured by a robot during the 100 time steps).

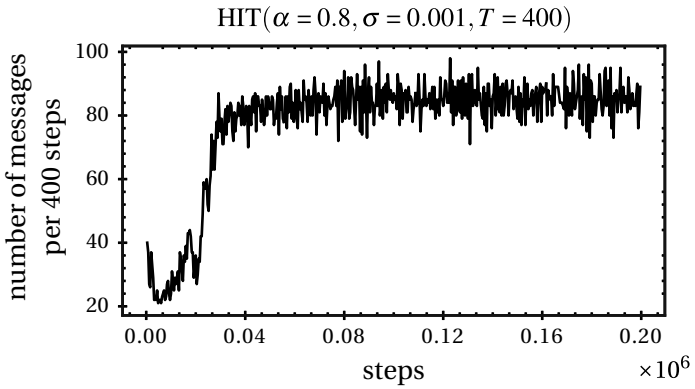


Figure 4.10: Evolution of the frequency of messages for one experiment, $\alpha = 0.8, \sigma = 10^{-3}$, $T = 400$. We count the number of messages sent in a constant interval of 400 iterations and report this value as a function of the step number

meta-parameters. We measure the characteristic time and the final fitness the policies obtained by the HIT algorithm, for a large combination of α and σ values.

We explore α values from 0.2 to 1.0 with a step size of 0.1, and σ from 10^{-8} to 10^{-2}, and 0. This represents a total of 18432 independent runs (9×8 combinations, 256 replicates per combination)[II]. In order to minimise to computational cost, we use Perceptron with no hidden layer, but with additional sensory inputs that do not bring useful information.

Results are shown in Figures 4.12 (final fitness) and 4.13 (characteristic time). We can observe that the average fitness plateaus for mutation rates with $\sigma < 10^{-2}$ and a transfer rate with $\alpha \leq 0.9$. Large mutation rate of $\sigma > 0.01$ as well as the maximum value for transfer rate $\alpha = 1$ are both destructive, whatever the other meta-parameter value. Regarding mutation, it is expected that a too large mutation rate can disrupt selection, and injects noise rather diversity that can be exploited. As for transfer rate, setting $\alpha = 1$ limits convergence to full control parameter sets present in the initial population only, as it is does not allow to benefit from recombination.

Aside from these extremes, HIT is revealed to be remarkably robust in terms of the final fitness that is reached. The transfer rate appears as the main parameter to modulate the convergence speed, with mutation being either destructive in the worst case $\sigma \geq 10^{-2}$, or of limited interest. In terms of efficiency of policies, the algorithm is rather robust with respect to its meta-parameter values, as long as extreme values are

[II]Experiments took 12 days using an Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz

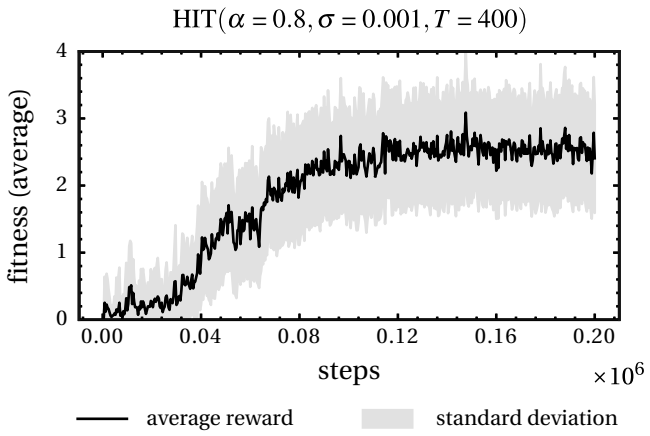


Figure 4.11: Evolution of the average fitness among 150 agents for one experiment of foraging 100 objects, $\alpha = 0.8, \sigma = 10^{-3}, T = 400$. At every encounter, the best agent share eighty percent of it randomly chosen parameters to the other agent. The worst receives these parameters and use them to improve it own policy. This process can only happen after both agents have spend at least 400 steps to evaluate themselves. During the maturation period, the fitness before modification of the best is reported for the averaging. No mutations are used here.

avoided ($\alpha > 0.9$ and $\sigma \geq 10^{-2}$).

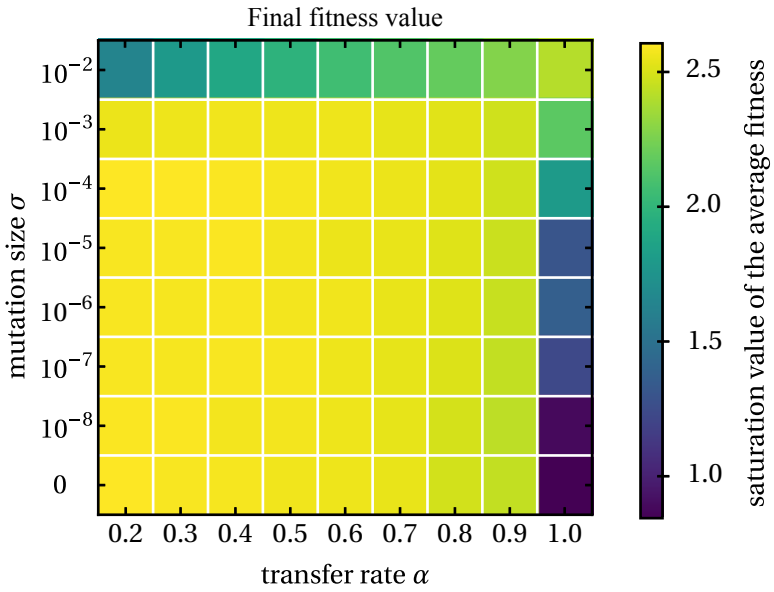


Figure 4.12: Final value of the average fitness for various mutation size and transfer rate. This value is the average of all fitness on the last 100 measurements (one measurement per 400 iterations). 256 independent simulation run have been used. Lighter colour means better saturation value.

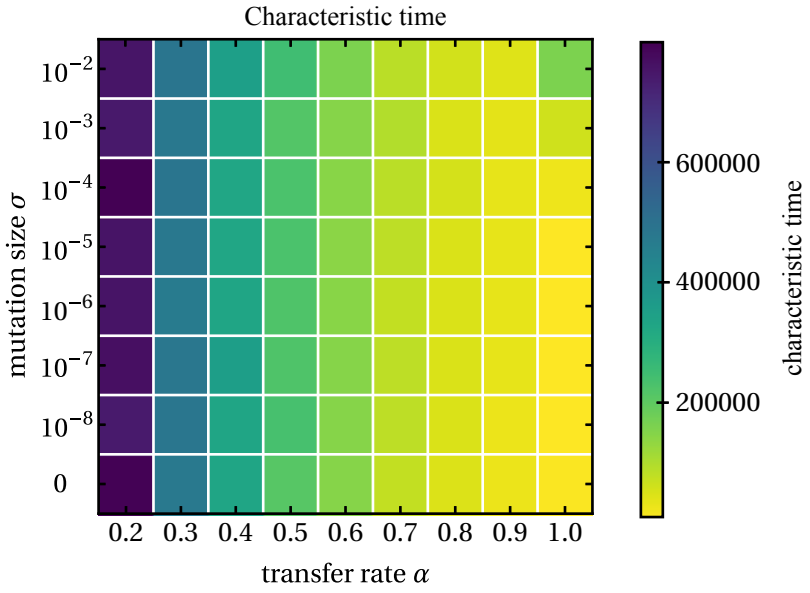


Figure 4.13: Characteristic time for various mutation size and transfer rate. The characteristic time is the step at which the average fitness reach half of it final value. 256 independent simulations run have been used. Lighter colour means shorter convergence time.

4.4.3 Learning the Transfer Rate

As stated at the beginning of this chapter, our aim is to provide an algorithm that can cope with limited communication capabilities. However, it is hardly possible to know in advance what are the limits. Technical specifications provide a good start, but the impact of robots density on packet collisions or perturbations from other sources makes it difficult to set an appropriate transfer rate beforehand.

Here, we study the HIT algorithm in a pseudo-realistic context where communication is artificially limited. In this experimental setting, all messages larger than $m \times 0.6 \times$ dim(parameter), are lost. The optimal transfer rate is therefore $\alpha_{opt} = 0.6$, but is unknown before deployment[III].

Resilience to communication failure is addressed by implementing a straight-forward learning method for the transfer rate. The transfer rate value is incorporated in the parameter set of each robot, and initially set to a random value $\alpha \in [0,1]$. It is then tuned by selection pressure.

[III] We also tested for $\alpha = 0.3$ with identical results (not shown here).

We performed 64 experiments, setting a mutation rate to $\sigma = 0.001$ only for the parameter α and $T = 400$ (as earlier). For *all* experiments, the whole robot swarm systematically converged to an α value close to the optimal transfer rate as shown on Figure 4.14 (left).

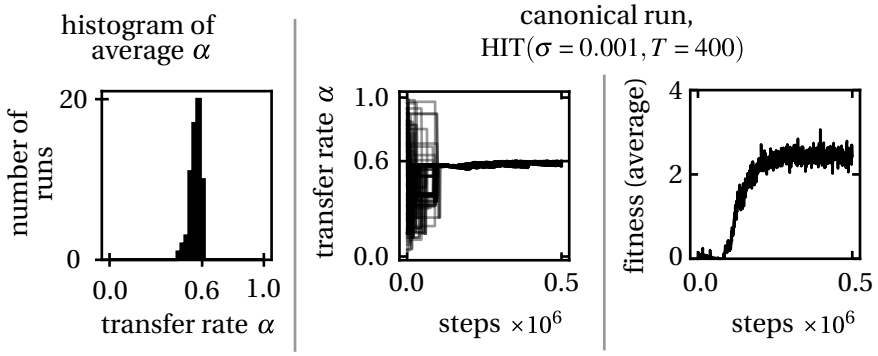


Figure 4.14: **Left**: distribution on 64 independents runs of the average transfer rate α after 5×10^5 iterations, with $\alpha_{opt} = 0.6$. **Center** and **Right**: learning the transfer rate α in a canonical run. Center: each line represent the trajectory of the α value for one specific agent. Right: average fitness for the population until convergence.

This is illustrated in Figure 4.14 (center), which represents the distribution of all α from the 150 robots across time steps for a typical run. As expected, α values are initially uniformly distributed between 0 and 1, and then quickly converge to a value close to α_{opt}.

Robots carrying α values which are too large with respect to the communication constraints simply cannot spread, and can only survive as long as it takes to encounter a robot with better fitness and capable of transmitting its own parameters. On the other hand, robots with small values of α can spread, but may do so at a smaller frequency than robots with relevant α values. This can be viewed as a lexicographic selection: the better performing robot will always fare better, and will diffuse faster if its α value makes the best of the environment at hand.

From a practical viewpoint, and extrapolating also from results in the previous section, it means that learning the transfer rate is always a good idea as long as it is limited to be strictly inferior to 1.

4.5 Conclusion

In this chapter, we introduced a new algorithm called *Horizontal Information Transfer*, which is at the crossroad of embodied evolution and social learning. We showed that this algorithm is competitive with the state of the art, but is also able to deal with limited communication capabilities that are often met with low-cost robots used in swarm robotics.

The obvious next step is to run full experiments with a swarm of low-cost robots. The implementation on robots (Kilobots) of the HIT algorithm has been performed and confirms the feasibility of the method to optimize robots after deployment (Mirhosseini et al., 2022; Zion et al., 2021).

5 Centralized Cooperative Co-Evolution with Adaptive Team Composition for a Resource Selection Problem

This chapter is adapted from:

- Fontbonne, Nicolas, Nicolas Maudet, and Nicolas Bredeche. "Cooperative Co-evolution and Adaptive Team Composition for a Multi-rover Resource Allocation Problem." *European Conference on Genetic Programming (Part of EvoStar)*. Springer, Cham, 2022.

5.1 Introduction

When multiple individuals get together to solve a task, it is sometimes difficult to identify who is actually contributing, and who is not. This is especially problematic when the benefits are equally shared among individuals, including with free-riders who invest a minimal amount of effort. Nature abounds from such examples and various strategies evolved to mitigate the detrimental cost of free-riding, such as partner choice or reputation (Noë and Hammerstein, 1994; West et al., 2007).

The problem of identifying the marginal contribution of individuals has also been studied extensively in cooperative game theory (Shoham and Leyton-Brown, 2008b). However, exact methods such as computing the Shapley value (Shapley, 1953b) require strong assumptions (e.g. ability to replay coalitions) and unrealistic computation time, which have led to a flourishing research into the design of approximate methods (Tumer et al., 2002; D. H. Wolpert and Tumer, 2001, 2008). The basic idea of such

methods is to identify the individual's contribution by computing the difference between the group performance with and without this very individual (e.g. by removing it or by replacing it with an individual with a default strategy).

In this research, we are interested in ad hoc autonomous agent teams where agents must act together without pre-coordination (Stone et al., 2010), which implies that the environment is non-stationary as all agents learn in parallel. This requires using methods that can only alter individuals' strategy, with neither a default strategy being known nor the possibility to remove temporarily one particular individual.

Such problem settings have been explored in evolutionary computation for multiagent systems, and notably with Cooperative Co-Evolutionary Algorithms (CCEA) which were first introduced in (Potter and De Jong, 1994b, 2000) and largely explored since then (see Ma et al., 2019 for a review of variants and applications). In particular, CCEA have been explored in setups involving multiple robotic agents in tasks such as exploration and foraging (Gomes et al., 2017) and environment monitoring (Rahmattalabi et al., 2016; Rockefeller et al., 2020; Zerbel and Tumer, 2020).

In this chapter, we address the problem of isolating team members so as to identify their contribution within the collective. On the one hand, one could allow only a single agent to learn at a given time, making it possible to measure its contribution accurately as other agents' strategies would remain stationary. On the other hand, several (or all) of the agents' policies could be iterated at the same time, possibly speeding up learning thanks to parallelization.

Balancing between providing accurate estimation of an individual contribution and parallelization of learning actually depends on the context at hand. When rewards are sparse and depend on a single individual's behavioural innovation, it is preferable to bootstrap learning with large-scale exploration. However, whenever team performance increases it is more efficient to turn towards a more conservative search so as to retain previous improvements. Finally, one less obvious situation arises when the synergy between individuals is required to improve performance, for example when two robots are required to open a door, none of which would gain any benefit by acting alone.

The rest of the chapter is organized as follows. Section 5.2 presents the problem of team composition in CCEA. Section 5.3 presents two CCEA algorithms that enable

to tune the number of learning agents within one learning step. The two algorithms differ with respect to how the balance between contribution estimation accuracy and learning parallelization is set: fixed, or self-adaptive. Section 5.4 presents the problem used for evaluation, which is a modified version of the famous Bar El Farol problem (Arthur, 1994) formulated as a multi-robot resource allocation problem where coordination is required (i.e. several resources must be harvested and harvesting is extremely beneficial if the optimal fixed number of robots is met). Results are presented in Section 5.5 for both the ad hoc version of and the self-adaptive versions of the algorithm.

5.2 Cooperative coevolution and team composition

In its most simple setup, cooperative co-evolutionary algorithms (CCEA) rely on a collection of independent evolutionary algorithms, with each dedicated to optimizing the policy of one particular agent of the team (De Jong, 2016; Eiben and Smith, 2003). Each independent algorithm works to improve the performance of one agent's control parameters using an assessment of the team performance.

Each algorithm maintains a population of parameter sets, which define candidate policies for the agent this algorithm is in charge of. At each generation, performance assessment is computed for various teams. Then, each instance of the CCEA tries to improve its agent's performance by using classic evolutionary operators of selection, mutation and recombination.

The problem faced by CCEA is thus a black-box optimization problem, with the additional twist that evaluation is for the *whole* team, and optimization is performed at the individual level, thus implying a weak link between team evaluation and the actual individual behaviour. Defining $\boldsymbol{\theta}$ as the vector containing the parameters provided by each algorithm of the CCEA, F as the fitness function used to assess team performance and f the fitness value, we have:

$$F : \text{team parameters } \boldsymbol{\theta} \longrightarrow \text{fitness value } f$$

Figure 5.1 illustrates the learning loop of a simple multi-agent black-box optimization procedure for cooperative co-evolution. A given algorithm in charge of a particular

agent i will provide policy parameters θ^i for this agent to be evaluated in a team. These parameters will be evaluated alongside parameters provided by the other agents. The team is then evaluated and a fitness value is returned.

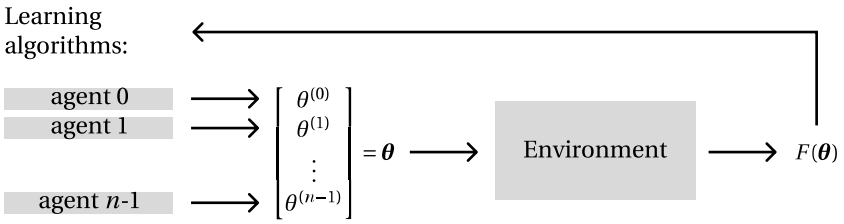


Figure 5.1: The learning loop of interaction. All agents submit their own parameters independently for evaluation. They are evaluated at the same time on the environment or task. This returns the fitness f of the whole team. Finally this feedback is used by all agents to update their parameters submission for next iteration.

After an evaluation, each agent has to evaluate if the new set of parameters proposed θ^i has contributed to the team fitness in a negative, positive or neutral manner. It is necessary to extract from the fitness $f = F(\boldsymbol{\theta})$, the part which depends only on the parameters of the agent i, $F^i(\theta^i)$.

From an agent's viewpoint, an increase in team performance may be due to others, and may even shadow a decrease in the very contribution the agent performs. In order to mitigate the intrinsically noisy fitness evaluation due to team heterogeneity (team composition changes over time), multiple evaluations of the same set of policy parameters will be performed for a given agent, so that different versions of θ^i can be ranked and further selected to create new candidate policies for the next generation. However, obtaining an exact assessment of one individual's contribution to the team remains elusive unless all other individuals follow static policies. Considering teams formed of n team agents, with each agent's evolutionary algorithm maintaining a population of p candidate policies, complexity would be $\mathcal{O}(p^n)$ at each generation.

In order to provide results in a reasonable time, CCEA generally relies on a partial evaluation of agents' policy contribution, by evaluating a subset of all possible team compositions at each iteration. Though such implementation breaks down complexity, CCEA algorithms have been shown to have a tendency to prematurely converge to stable states because of a deceptive fitness landscape created by the choice of collaborators for evaluation (Funes and Pujals, 2005; Panait, 2010). Several methods

have been proposed to address these issues, including novelty-based rewards to escape local minima (Gomes et al., 2017) or automatically merging populations when agents' behaviours are similar to address scalability issues (Gomes et al., 2018, 2019). However, the contribution of one specific agent remains approximated rather than precisely measured.

5.3 Cooperative Co-Evolutionary Algorithms with Limited Team Composition Update

In the simplest case, the global fitness $F(\boldsymbol{\theta})$ is the sum of each agent's individual fitness $F^i(\theta^i)$ for the current team:

$$F(\boldsymbol{\theta}) = \sum_{i \in \text{agents}} F^i(\theta^i)$$

With θ^i the policy parameters for agent i, and $\boldsymbol{\theta} = (\theta^0, \ldots, \theta^i, \ldots, \theta^{N-1})$, i.e. the team policy parameters composed of the policies of N agents.

Whenever a single agent updates its policy parameters, the variation in overall fitness $F(\boldsymbol{\theta}_{t+1}) - F(\boldsymbol{\theta}_t)$ will be equal to the variation in the fitness due to the change in behaviour of the agent concerned δF^i. This can be written as:

$$\begin{aligned} F(\boldsymbol{\theta}_{t+1}) - F(\boldsymbol{\theta}_t) &= F^{-i}(\theta^{-i}_{t+1}) + F^i(\theta^i_{t+1}) - F^{-i}(\theta^{-i}_t) - F^i(\theta^i_t) \\ &= F^i(\theta^i_{t+1}) - F^i(\theta^i_t) \\ &= \delta F^i \end{aligned} \tag{5.1}$$

With $F^{-i}(\theta^{-i})$ the performance of all individuals minus the agent i, assuming θ^{-i} is stationary between t and $t+1$. Though an exact value for the contribution of agent i remains unavailable, δF^i gives a proxy which provides sufficient information to measure both the direction and amplitude of the change due to agent i's new policy.

Assuming agents are independent, the above equation holds true and can be used as long as only one agent's policy is changed at a time. However, this assumption incurs two important limitations:

- The computational cost of iterating over a single agent's strategy at a time is high (see previous section), and there is a trade-off between the quality of one agent's contribution estimation and the expected gain at the level of the team (e.g. whenever a single robot is needed to significantly improve team performance, trying with all robots is relevant);
- The task may require coupling between the agents' behaviour, and any team fitness improvements may require that *several* agents change their policy parameters simultaneously (e.g. moving a heavy object can only be done with two robots). If not, an algorithm can get stuck on a local minimum if we assume independence between agents and change only one agent at a time.

In order to address these issues and still retain the benefit of accurate estimation of the agents' contribution, we propose a CCEA algorithm where it is possible to modulate the number of agents that are updated in-between team evaluations. We use a collection of (1+1)-GA algorithm where each (1+1)-GA algorithm i provides the policy parameters θ^i for its corresponding agent i, and the whole team is evaluated using all agents with their policy parameters, i.e. $\boldsymbol{\theta} = (\theta^0, \ldots, \theta^{N-1})$.

Algorithm 3: CC-(1+1)-GA$_{k_{\text{fixed}}}$
Introducing k mutants per iteration

$k \leftarrow$ number of team members to be updated;
$N \leftarrow$ total number of agents;
$\boldsymbol{\theta}_{\text{parent}} \leftarrow$ parameters initialisation;
$f_{\text{parent}} \leftarrow F(\boldsymbol{\theta}_{\text{parent}})$;
for *gen* < *nb max generation* **do**
    ID $\leftarrow$ randomly sample k agents;
    $\theta^{ID}_{\text{child}} \leftarrow \text{mutate}(\theta^{ID}_{\text{parent}})$;
    $f_{\text{child}} \leftarrow F(\theta^{ID}_{\text{child}}, \theta^{-ID}_{\text{parent}})$;
    if $f_{child} \geq f_{parent}$ **then**
        $\theta^{ID}_{\text{parent}} \leftarrow \theta^{ID}_{\text{child}}$;
        $f_{\text{parent}} \leftarrow f_{\text{child}}$;
    end
end

Algorithm 3 details the complete CCEA, which runs multiple instance of (1+1)-GA in parallel, which we will refer to as the CC-(1+1)-GA$_{k_{\text{fixed}}}$ algorithm from now on. Each (1+1)-GA algorithm maintains a population of two individuals (Beyer and Schwefel, 2002), a parent θ^i_{parent} and a child θ^i_{child}. Both are candidate policy parameters for

agent i. The parent is replaced when the child fares better, and a new child is created by applying mutation on the new parent. Whenever a child fails to outperform its parent, it is replaced by a new child mutated from the current parent. The mutation operator depends on the problem (e.g. Gaussian mutation, bit-flip, uniform draw).

At each new iteration, k agents are drawn and randomly changed in the team, with $0 < k \leq N$. The k parameter is used to tune the amount of renewal k/N for the team composition in-between iterations of the CCEA algorithm. The k new team members are kept only if they provide an increase in the team fitness. Therefore, the challenge is to find the most efficient size k of the number of team agents to be modified at each CCEA steps. So far, k is fixed beforehand by the user, and may benefits from prior knowledge on the task regarding possible required coupling between agents, in terms of number of agents to change simultaneously to reach the global optimum in terms of team efficiency.

However, such prior knowledge may not be available and a relevant value of k not only depends on the problem (e.g. some problems may require coupling between agents, others may not), but also on the current state of the optimization (e.g. broad initial search steps vs. refined tuning near the optimal solution). In order to address this, we propose the CC-(1+1)-GA$_{k_{\text{adaptive}}}$, where the k parameter is learned during the course of evolution (see Algorithm 4). We propose to choose the number of team members to be updated by using the adversarial bandit learning algorithm EXP3 (Exponential-weight algorithm for Exploration and Exploitation (Auer et al., 2002)) that tracks the success rate of various possible values for k so far, which means both exploiting the current best value and exploring alternate values. The goal of the adaptation mechanism is to converge to the best possible value for k for the context at hand, i.e. the value that leads to the largest increase of fitness, whether through rare but important increases or through small but frequent increases.

As described in Algorithm 4, we define a set of J possible values for k $(k_0, ..., k_{J-1})$, each associated with a weight $W(k_j)$ monitoring the success rate of a particular k_j. Lines 10-13 of the algorithm detail which k_j is selected for a particular iteration. We compute the probability distribution of each k_j which depends on the weight $W(k_j)$ and the parameter γ of the algorithm. $\gamma \to 1$ favours exploration (i.e. the choice of k_j will be almost uniform). On the contrary, $\gamma \to 0$ favours exploitation, taken into account the importance of the weights $W(k_j)$. The fitness gain is normalized between

$[0,1]$ (Line 20) and then used to update the weight $W(k_j)$ (Line 21).

Algorithm 4: CC-(1+1)-GA$_{k_{\text{adaptive}}}$
Replacing a varying number of team agents per iteration

1 $K \leftarrow$ table of possible number of team members to update simultaneously ;
2 $W \leftarrow$ table of weights for each k;
3 $P \leftarrow$ table of probability for each k;
4 $k \leftarrow$ number of team members to be updated ;
5 $N \leftarrow$ total number of agents;
6 $\boldsymbol{\theta}_{\text{parent}} \leftarrow$ parameters initialization;
7 $\gamma \leftarrow$ real $\in [0,1]$, parameter for the EXP3 algorithm ;
8 $f_{\text{parent}} \leftarrow F(\boldsymbol{\theta}_{\text{parent}})$;
9 **for** *gen* < *nb max generation* **do**
10 **for** $j = 1, \ldots, J$ **do**
11 $P[j] \leftarrow (1-\gamma)\frac{W[j]}{\sum_{i=1}^{J} W[i]} + \frac{\gamma}{J}$
12 **end**
13 $k_j \leftarrow$ random draw in $K[]$ with probabilities $P[]$;
14 ID $\leftarrow$ randomly sample k agents;
15 $\theta^{ID}_{\text{child}} \leftarrow \text{mutate}(\theta^{ID}_{\text{parent}})$;
16 $f_{\text{child}} \leftarrow F(\theta^{ID}_{\text{child}}, \theta^{-ID}_{\text{parent}})$;
17 **if** $f_{child} \geq f_{parent}$ **then**
18 $\theta^{ID}_{\text{parent}} \leftarrow \theta^{ID}_{\text{child}}$;
19 $f_{\text{parent}} \leftarrow f_{\text{child}}$;
20 $R \leftarrow \tanh\left(\frac{f_{\text{child}}}{f_{\text{parent}}}\right)$;
21 $W[j] \leftarrow W[j] \exp\left(\frac{\gamma R}{P[j]J}\right)$;
22 **end**
23 **end**

The whole algorithm loop is also represented in Figure 5.2. In this example, five agents are learning. The current k is 2, meaning that, for this generation, two agents are mutated, giving birth to two children, and are evaluated with the others parents.

5.4 The Multi-Rover Resource Selection Problem

We define a problem that is a variation of the well-known El Farol bar problem (Arthur, 1994; D. H. Wolpert and Tumer, 2008) where each individual must choose a day to go to the bar among M possible choices with the criterion of not being too numerous each days. In our setup, we consider the problem where N independent robots must

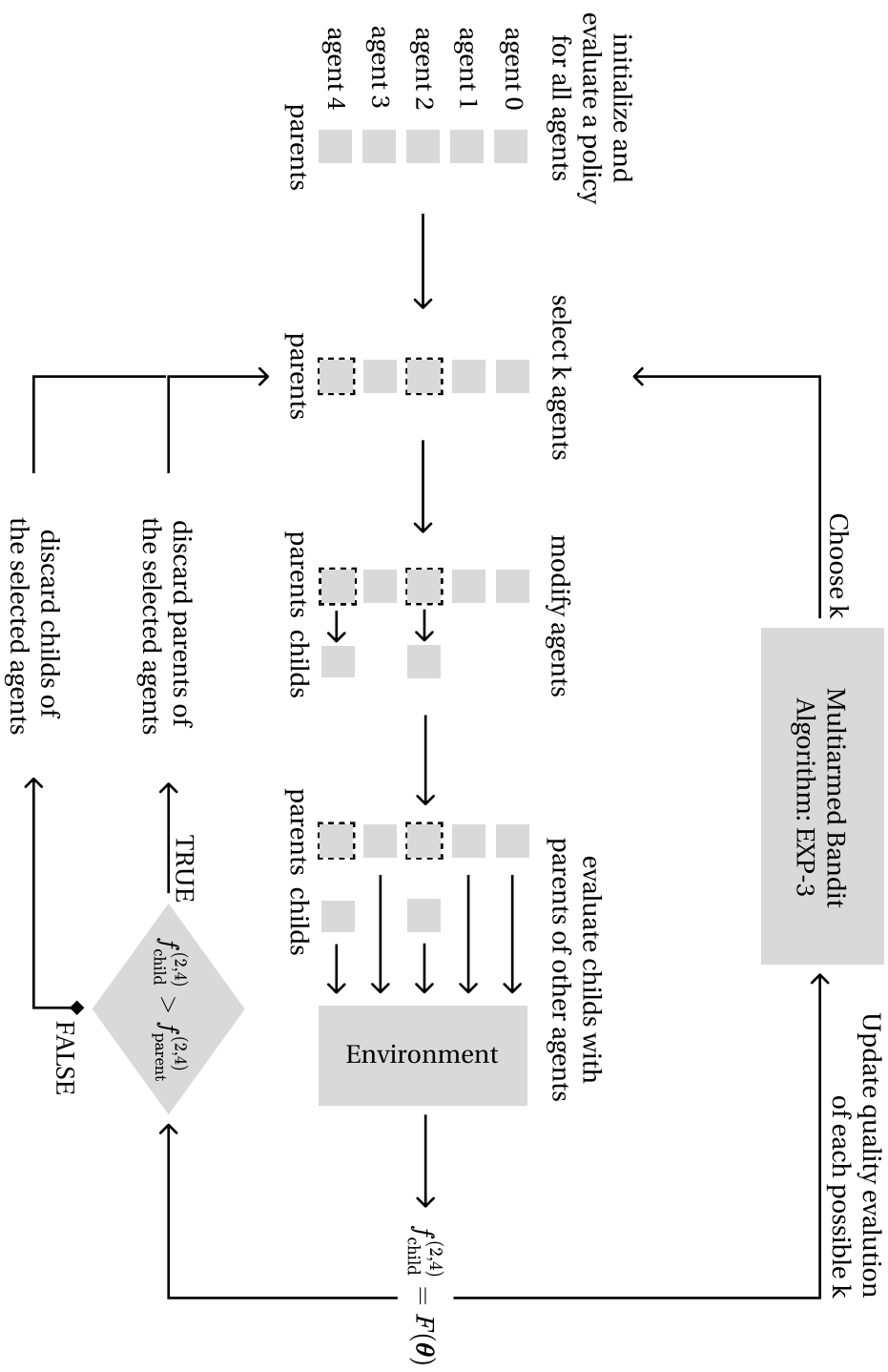
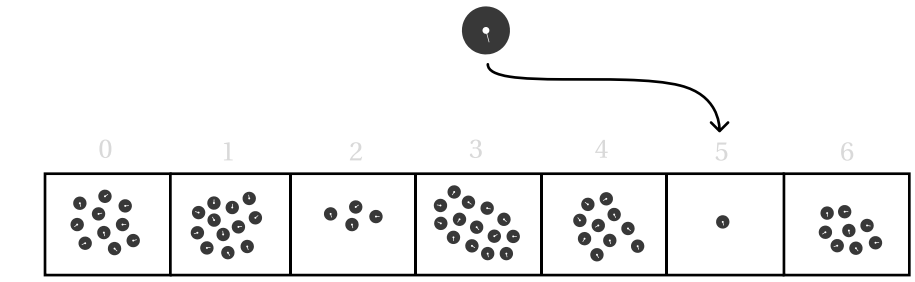


Figure 5.2: Diagram of the different steps of the CC-(1+1)-GA$_{k_{\text{adaptive}}}$ algorithm.

spread over M resources, and where each resource has an optimal capacity c in terms of number of robots necessary to optimally harvest it. This is illustrated in Figure 5.3,

which provides an example where each robot chooses a resource.



$$f = \phi_c(10) + \phi_c(12) + \phi_c(4) + \phi_c(14) + \phi_c(10) + \phi_c(2) + \phi_c(8)$$
$$= 66.483$$

Figure 5.3: The $N = 60$ robots are represented here as little rovers that each must choose between $M = 7$ resources. Here the selected agent chooses resource 5. When all robots have made their choices, the satisfaction for each resources are computed and summed to obtain the fitness f of the team.

Team performance f is obtained by adding each resource's satisfaction ϕ_c. For each resource, its satisfaction ϕ_c depends on the number of robots r who chose it. This satisfaction is described by the following equation:

$$\phi_c(r) = \begin{cases} Mr\exp(\frac{-r}{c}) & \text{if } r = c \\ r\exp(\frac{-r}{c}) & \text{else.} \end{cases} \tag{5.2}$$

where r represents the amount of robots on the resource, M the total number of resources, and c controls the optimal number of robots required for the resource.

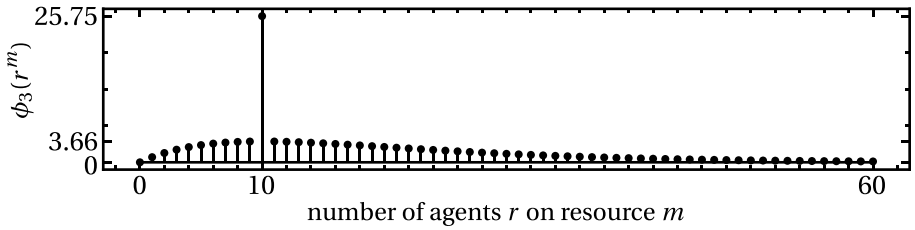


Figure 5.4: Satisfaction function with $c = 10$ of a given resource, depending on the number of robots that picked it

The satisfaction function diverges from the original formulation of the El Farol Bar problem as the best team performance always implies that the number of robots per resource must be optimal (exactly c robots per resource), even if it implies some

resources are left aside or only partially filled. The satisfaction boost for the $r = c$ case ensures that filling a maximum number of resources with the c robots is the optimal strategy. An example of such function with $c = 10$ is plotted on figure 5.4.

The fitness of a team is then the sum of the satisfaction for all resources:

$$f = \sum_{m \in [0, M-1]} \phi_c(r^m)$$

Where r^m is the number of robots at resource m. The robots must coordinate to optimally fill a maximum number of resources.

To increase the value of this function, it is then necessary to move individuals from crowded resources to resources with fewer robots, up to the extent that resources with the exact number of robots are favoured.

The number of robots N, the number of resources M, and the optimal number of robots per resource c can be modified to change the structure of the problem. In the next section, different instances of this problem are used to study various properties of the algorithms we proposed in the previous section. In particular, it is possible to set up the problem so that either single or multiple changes in the team composition may always yield too few or too many permutations in the team distribution over resources for team performance to increase.

5.5 Results

5.5.1 Experimental setting

We use the Multi-Rover Resource Selection problem, with different number of resources M, number of agents N, and optimal number of robots per resource c. The three setups used are:

- **Setup 1** with $N = 120$, $M = 300$, and $c = 30$. There are many resources, each requiring a large number of robots. The maximum performance could be reached by a team of exactly $M \times c = 300 \times 30 = 9000$ agents. Given the limited number of agents, they must spread over a few of the resources ($N/c = 120/30 = 4$ resources) so that the team reaches optimal performance;

- **Setup 2** with $N = 900$, $M = 300$, and $c = 3$. The number of robots involved makes it possible to reach the optimal team performance value for this setup ($N = M \times c$) if all agents are uniformly spread over the resources;
- **Setup 3** with $N = 60$, $M = 7$, and $c = 10$. There exists several configurations of pairing agents and resources which are local optima *and* cannot be escaped by updating only one agent in the team. Figure 5.5 gives an example of a sub-optimal configuration for which using $k = 1$ is detrimental. When the algorithm gets into such a configuration, all possible updates of a unique agent will decrease the team fitness. Escaping such a local optimum requires either exploring new configurations at the cost of a (hopefully temporary) decrease in team performance (see Gomes et al., 2017 for example using novelty search in CCEA, which is out-of-scope of the current chapter) or modifying several agents at the same time (which is possible with $k > 1$).

In the following, we use both the CC-(1+1)-GA$_{k_{\text{fixed}}}$ algorithm with either $k = 1$, 10 or 30, and the CC-(1+1)-GA$_{k_{\text{adaptive}}}$ algorithm (using EXP3) for learning the value of k in $\{1, 10, 30\}$. All experiments are replicated 32 times. Mean and standard deviation for all algorithm variants are traced. Evaluations is used on the x-axis to provide a fair comparison in terms of computational effort.

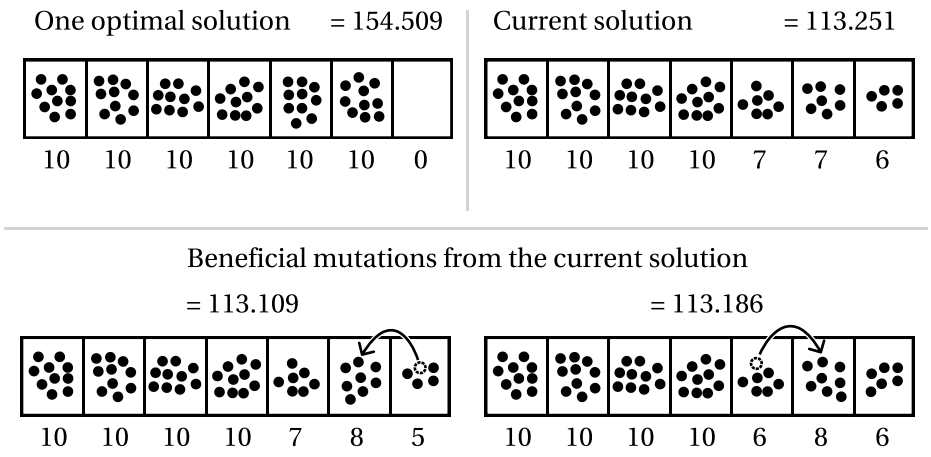


Figure 5.5: The resource selection problem has local minimums that can't be escaped by mutating only one agent. In this example, 60 agents must spread on 7 resources by being 10 on 6 of them. In the state where 4 resources are selected by 10 agents, 2 are selected by 7, and 1 by 6, modifying the selection of one agent can only decrease the fitness of the system.

5.5.2 Fixed vs. Adaptive Methods for Team Composition Update

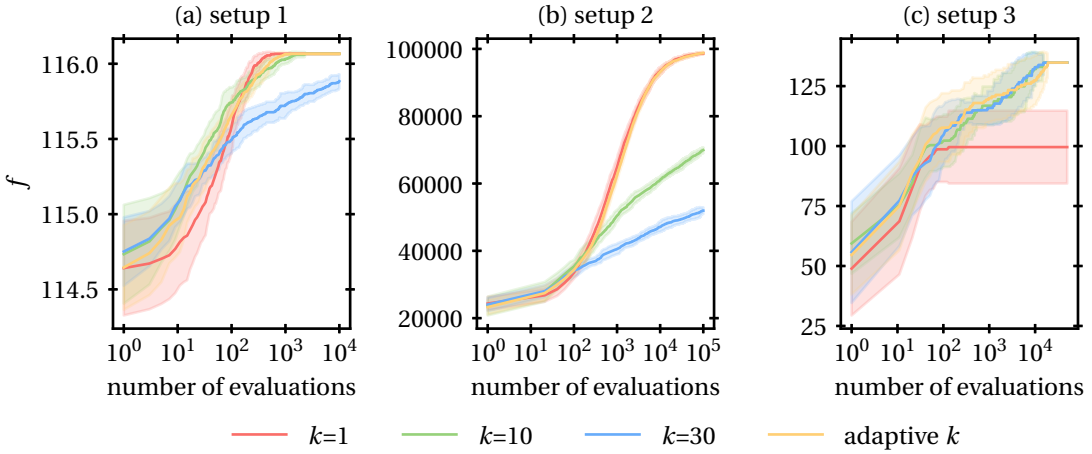


Figure 5.6: Performance of the CC-(1+1)-GA$_{k_{\text{fixed}}}$ algorithm with either $k = 1$, $k = 10$ and $k = 30$, as well as CC-(1+1)-GA$_{k_{\text{adaptive}}}$ with $k \in \{1, 10, 30\}$. Performance f is plotted as mean (solid lines) and standard deviation for the three setups considered with respect to the number of fitness evaluations. Curves are plotted on a x-log scale. There are 32 replications for each algorithm and for each experiment.

Starting with the three variants of the CC-(1+1)-GA$_{k_{\text{fixed}}}$ algorithm, we can observe different learning dynamics depending both on the value of k and the setup at hand.

In the first setup (Fig. 5.6(a)), we observe a clear benefit for using $k = 10$ and $k = 30$ during the first iterations. But this initial gain in performance does not allow it to converge faster when using $k = 1$. In particular, a value of $k = 30$ is extremely deleterious for the convergence as it fails to reach the optimum value within the allocated evaluation budget. This tendency is even more visible in the second setup (Fig. 5.6(b)). Using larger values for k provides a slight advantage at the beginning but is quickly lost for both $k = 10$ and $k = 30$.

The outcome is rather different in the third setup (Fig. 5.6(c)) as using $k > 1$ allows to reach better performances and prevent the algorithm to get stuck on a local optimum. Indeed, the algorithm becomes stalled when using $k = 1$, the structure of the problem making it impossible to improve team performance without considering coupled synergies when updating team members.

Figures 5.6(a) and (b) show that the CC-(1+1)-GA$_{k_{\text{adaptive}}}$ algorithm follows the curves of the best performing algorithms using a fixed value of k. Figure 5.6(c) also shows

that the adaptive algorithm is able to adapt to a situation where the CC-(1+1)-GA$_{k_{\text{fixed}}}$ algorithm would fail because of its fixed k value (here, using $k = 1$ withholds convergence to an optimal team composition). Overall, dynamically modulating the number of policies updated in the team composition always results in performance curves closely matching the best out of the algorithmic variants using a fixed value of k. This remains true even when the best variant with a fixed k value is outperformed by another variant with a different value of k, which confirms the relevance of the adaptive algorithm to act as an anytime learning algorithm. In other words, the CC-(1+1)-GA$_{k_{\text{adaptive}}}$ algorithm presents the best choice when the problem and the evaluation budget are not known.

5.5.3 Dynamics of Adapting the Number of Team Agents to Update

We analyze how the CC-(1+1)-GA$_{k_{\text{adaptive}}}$ algorithm is changing the value of k throughout evolution for the three setups at hand. Figure 5.7 represents the median value of k over the 32 repetitions for each of the three experimental setups. We observe that the algorithm switches between the different values for k, and follows different dynamics depending on the setup.

In the first setup, the method slightly favours $k = 1$ and $k = 10$ after a few iterations of exploration. This bias is consistent with the performances observed for k fixed, where the $k = 30$ version is less efficient (see Fig. 5.6). In the second setup, the value of k decreases during the learning process to stabilize at $k = 1$, allowing for some fine-tuning of team composition. The third setup displays somewhat different dynamics

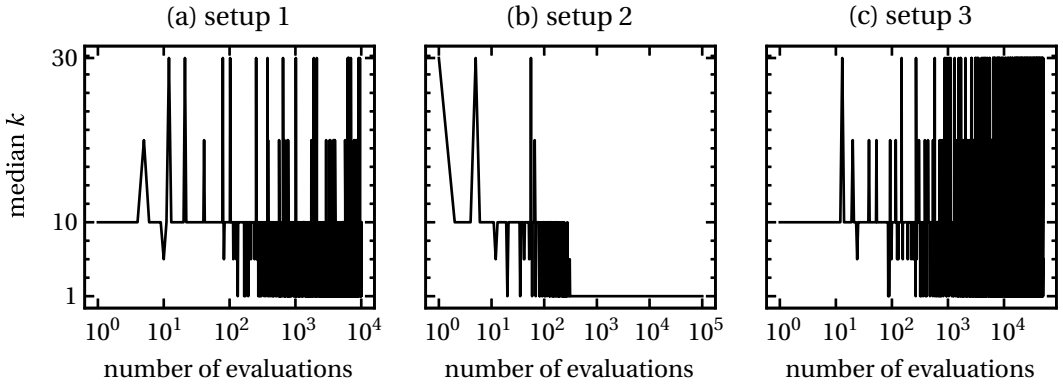


Figure 5.7: Median value of k over the 32 repetitions for the first (left), second (middle) and third setups (right). Curves are plotted on a x-log scale.

for the value of k, quickly switching from one value to the other. The difference in performance between the different group sizes is not large enough to make a clear-cut choice, and the method chooses k uniformly at each step without impacting the performances.

5.5.4 Sensitivity of meta-parameters

As described in Algorithm 4, CC-(1+1)-GA$_{k_{\text{adaptive}}}$ uses two meta-parameters, which are:

- $K = (k_0, ..., k_{J-1})$, the set of possible values for k;
- the egalitarianism factor γ that determines at each step whether k should be chosen at random (uniform sampling), or selected with respect to the weights of the k values, obtained from the cumulative fitness gain for each particular value of k. The value of γ balances between exploitation and exploration, and the EXP3 algorithm for multi-armed bandit problems has been extensively studied elsewhere (Auer et al., 2002; Sutton and Barto, 2018);

In the previous section, these meta-parameters were fixed as follow: the set of possible k was limited to $\{1, 10, 30\}$, the egalitarianism factor γ was set to 0.1.

Figure 5.8 shows the sensitivity of the algorithm with respect to the different meta-parameters. From top to bottom, the sensitivity to γ and the set of possible k. The general conclusion from this study is that the algorithm is robust and remains a relevant choice for anytime learning. Learning curves remain close to what has been shown previously, with some exceptions for extreme values. In particular, we can observe that:

- γ does not have a significant impact on the algorithm, provided that it is not too small nor too high to efficiently modulate the exploration and exploitation of k's
- the algorithm is somewhat also sensitive to the cardinal of the set of possible values for k. When there are too many possibilities to explore, the evaluation of each choice takes more time and is, therefore, less accurate if the context changes too fast. The effect of this exploration can especially be observed for the second setup where the algorithm is less accurate when the value for k can be chosen among 30 possible values ($k \in [1, 30]$).

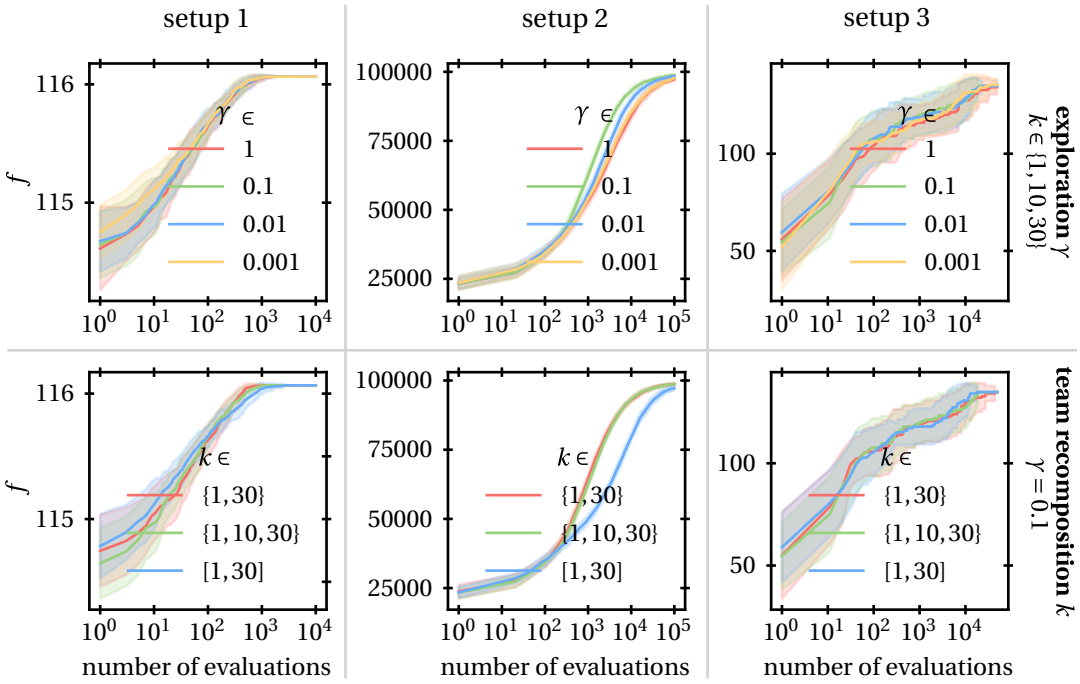


Figure 5.8: Sensitivity of the algorithm to meta-parameters. Each column represents one of the different experimental setups. On the rows, one of the meta-parameter is fixed and the other one is varying. On the top, the set of k are fixed but γ varies. At the bottom, γ is fixed but the set of k varies.

5.6 Conclusion

In this chapter, we presented a cooperative co-evolutionary algorithm (CCEA) that implements a collection of (1+1)-GA algorithms, each endowed with the task to optimize the policy parameters of a specific agent while performance is assessed at the level of the team. Our algorithm acts on team composition by continuously updating a limited number of team agents, depending on the task at hand and the level of completion. Therefore, the main contribution of this chapter is to describe an algorithm with a self-adapting team composition update mechanism used throughout learning.

We showed that modulating through time the number of new policies added to the current team makes it possible to provide efficient anytime learning, without requiring *a priori* knowledge on the problem to be solved. Moreover, we show that the algorithm can deal with problems where coupling between agents during learning is mandatory to improve team performance.

Experimental validation was conducted using a variant of the El Farol Bar problem, a famous problem in collective decision making, which was modified to capture a multi-agent resource selection problem. Our algorithm is indeed also relevant for multi-robotic setups, which have been recently studied using various CCEA algorithms (Gomes et al., 2017, 2018, 2019; Rahmattalabi et al., 2016; Rockefeller et al., 2020; Zerbel and Tumer, 2020). The next chapter will tackle such setups.

6 Centralized Cooperative Co-Evolution with Adaptive Team Composition for Pseudo-Realistic Multi-Rover Spatial Distribution Task

6.1 Introduction

The previous chapter introduced a cooperative co-evolution algorithm that modulates the number of agents exploring at each generation. However, the application domain was a simplified setup where agents had only one action to choose at each round. In this chapter, we analyze the performance of an extension of this algorithm on a more classical domain where the agents have to learn a complete deterministic policy that maps each observation to a continuous action domain.

Using more complex policies raises the question of the genetic composition of the team. Genetic composition refers to the relationship between the control policy parameters of the agents. We can thus have:

- Homogeneous team: all agents use the same control parameters, which means they would behave similarly under similar conditions. This can also be referred to as the clonal approach;
- Heterogeneous team: all agents use different control parameter values, which means that one agent is encoded by a unique set of control parameter values.

Homogeneous teams have been found to be scalable (Baray, 1997; Trianni and Dorigo,

2006), fast to evaluate (Luke et al., 1997), and still able to specialize (Bryant and Miikkulainen, 2003). While heterogeneous teams allow more flexibility in terms of role specialization (Baldassarre et al., 2003; Bernard et al., 2016; Bongard and Paul, 2000; Quinn et al., 2002).

More recent results refine these findings. Ferrante et al., 2015 showed that role specialization could be learned even within homogeneous teams as robots can adapt their roles depending on the environment. However, in some extreme cases requiring specialization, such as when leader/follower roles are required, heterogeneous teams are mandatory to get a stable organization between agents (Bernard et al., 2016).

Moreover, the multi-robots aspect of a genetically homogeneous system can be debated. In this case, all agents are clones, and the learning process consists of optimizing a single policy that is replicated on all agents. One could argue that this leads to a single agent system whose body is composed of multiple physically independent parts. On the contrary, using heterogeneous teams leads to more complicated optimization problems but allows the autonomy of each agent, which may lead to a more robust policy (Stone et al., 2010).

As emphasized by Waibel et al., 2009, the notion of genetic team composition is also related to the level of selection performed during the evolutionary algorithm. The two possible levels of selection are:

- Individual selection: each robot interact with a team but gets an individual evaluation. Control parameter values of the robots with the highest scores are more likely to be selected to build a new population of candidate solutions, whether they were initially part of the same team or not.
- Team selection: performance is evaluated for the whole team. No distinction is made between the participating agents, whether they are similar or not. Regardless of their individual contribution to the team's performance, all agents that were part of the best teams are more likely to participate in the construction of a new population of candidate solutions.

Figure 6.1 represents the four evolutionary conditions, combining the genetic composition of the team and level of selection.

Team selection guarantees agents will collaborate as they are selected only if they

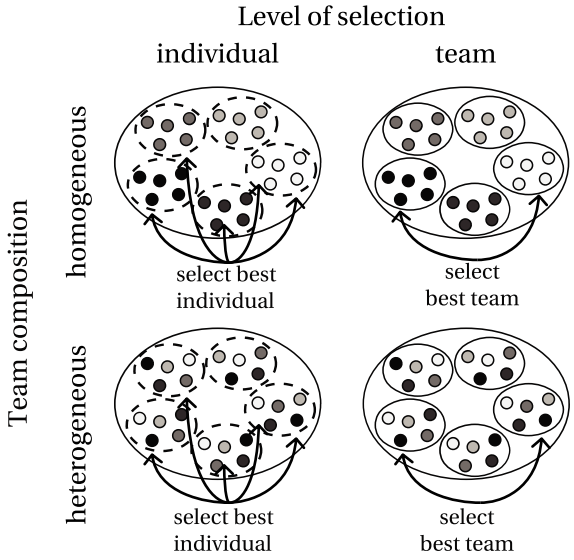


Figure 6.1: Four different evolutionary scenario. The population, represented by a large oval, is made up of several teams, represented by medium ovals. Each team is composed of several robots, represented by small circles, which are evaluated together. The genetic composition of the teams can be either homogeneous, with all the robots having the same genome and shown with identical shading, or heterogeneous, with the robots having different genomes and shown with different shading. The level of selection can either be at the team level, where entire teams are selected, or at the individual level, where individuals are selected independently of their team affiliation. Figure is reproduced from Waibel et al., 2009.

allow the whole team to obtain a better evaluation. However, this can be problematic in the case of heterogeneous team evolution. For example, by not taking into account the individual contributions of each agent, this method can encourage free riders. These agents benefit from the performance of their teammates to be selected, and this can be detrimental to the optimization process. This is especially true for large teams where the dilution of responsibility hides individual impacts.

On the other hand, individual selection allows for a more tailored evaluation for each agent. They are selected only if their performance is beneficial. This allows for more stability in the learning process as the selection is specifically tailored to one's individual performance.

In the case of a heterogeneous population, global fitness or team-level selection is rarely treated. However, these concern many problems that require the specialization of agents (i.e. they cannot have identical behaviours). They include construction tasks

(Nitschke, 2012), foraging of multiple objects (Tarapore et al., 2006), or herding (Potter et al., 2001). These problems are tackled by cooperative co-evolution algorithms (CCEA)(Potter and De Jong, 1994a). As described in 3.5.2, cooperative co-evolutionary algorithms evolve multiple populations of solutions simultaneously. These populations are called sub-populations, and are evolved cooperatively, in the sense that the fitness of each solution in a sub-population depends on the other sub-populations.

6.1.1 Related work

Independent learners suffer from the alter-exploration problem where agents fail to consider the exploration of their teammates that concurrently learn and update their policies. Sharing information on a network or through a centralized authority allows for mitigating this problem. Multiple solutions have been proposed. HolmesParker et al., 2014 introduce a reward structure to remove teammates' exploration noise from each agent's reward signal. It allows agents to explore without taking explicit actions. Instead, they calculate a counterfactual reward for an action they did not take, using a model of the system's reward function and the CLEAN shaping structure. The first way to do this is to compare the evaluation if we replace the current action by a new exploratory counterfactual action a'_i:

$$c1_i(\boldsymbol{a}) = r\left(\boldsymbol{a}_{a_i \leftarrow a'_i}\right) - r(\boldsymbol{a}) \tag{6.1}$$

We thus find the opposite of the difference reward, which consisted in comparing the reward of the current action to a default counterfactual. By combining these two approaches, we obtain the second version of CLEAN reward. It consists in comparing an exploratory action a'_i to a default action a''_i:

$$c2_i(\boldsymbol{a}) = r\left(\boldsymbol{a}_{a_i \leftarrow a'_i}\right) - r\left(\boldsymbol{a}_{a_i \leftarrow a''_i}\right) \tag{6.2}$$

This calculation is used to update the agent's policy, and other agents see only the non-exploratory action. This means that exploration does not add noise to the system and promotes coordination among agents.

Similarly, Chung et al., 2018 propose a method to limit the number of agents that are learning across each epoch. Agent updates their policy with a probability that depends on its predicted impact.

6.2 Algorithm

In this new experimental setup, the optimization object is a multilayer neural network (see Figure 6.4). It is thus a question of modifying the weights of the network, represented by a vector θ. The class of evolutionary algorithms that handles this kind of problem is usually named Evolutionary Strategies (ES). We introduce the CC-1+1-ES algorithm as an extension of CC-1+1-GA described in Section 5.3.

The main algorithm is recalled in Figure 6.2. k agents are randomly selected and mutated to produce children. These children are evaluated with the parents of the non-mutated agents. If the fitness of the team is better, the children are kept. Otherwise, the originals are kept.

In the adaptive case, the group size k is modified at each generation using the EXP-3 algorithm. This method allows choosing among a set of possible choices for k, according to the performance gains they allowed. It uses a parameter γ, which allows modulating the exploration or the exploitation. A γ closer to 0 indicates more exploitation of the best-performing group size k, while a γ closer to 1 will allow more exploration and the group size k will be drawn almost uniformly.

The mutation operator used in these experiments differs from the previous chapter. It chooses, with probability p, a *Gaussian mutation* operator on all parameters of an agent and, with probability $1-p$, a *uniform mutation* operator of a subset of its parameters. The Gaussian mutation operator applies a perturbation centred on the current parameter value, with a variance σ:

$$\theta \leftarrow \mathcal{N}(\theta, \sigma)$$

And the uniform mutation operator on range $\mathcal{R}$ of a subset V_m of m randomly selected parameters is defined as:

$$\theta[i] \leftarrow \mathcal{U}(\mathcal{R}) \quad \forall i \in V_m$$

Gaussian mutations allow for small variations around the current solution and thus allow for small adjustments in behaviour. Uniform mutations, on the other hand, lead to more radical modifications of the policy and allow a wider exploration of the parameter space.

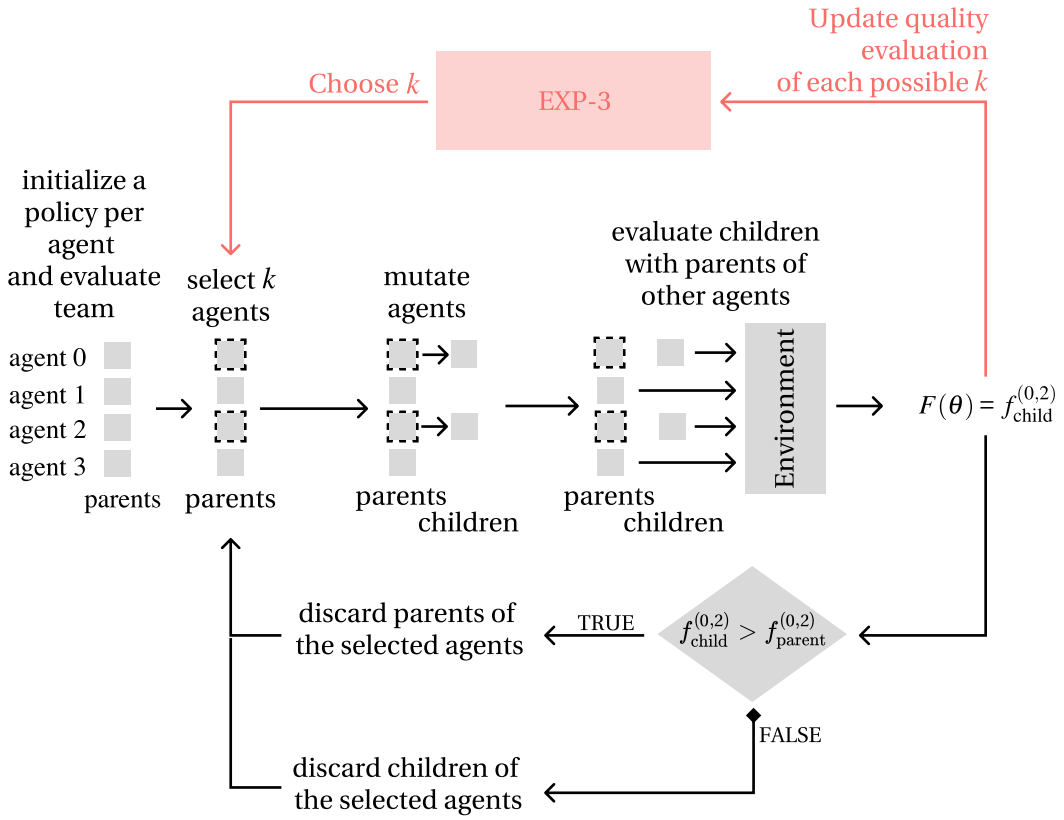


Figure 6.2: Diagram of the CC-1+1-ES algorithm. The top red part represents the choice of k for the k_{adaptive} version of the algorithm.

6.3 Experimental setup

6.3.1 Task and Environment

To study the dynamics of the algorithm in a new environment, we devise a rover exploration task inspired by Agogino and Tumer, 2004b (and further research, for example, in Dixit and Tumer, 2022; Dixit et al., 2019; Zerbel and Tumer, 2020). It is a good abstraction of a search and retrieves robotic task, where an unknown environment must be explored to retrieve specific objects or resources. The goal here is for a team of rovers to reach some points of interest (POI) that would give the most rewards. In our experiments, the rovers start in a small aggregate and must move in the environment sensing their teammates and points of interest.

More specifically, four robotic agents are placed in the center of an area with some

random variation of coordinates and orientation. These are detailed in Table 6.1 and represented by small coloured squares in Figure 6.3.

	x area	y area	orientation
agent 1	$[0, 0.5]$	$[0, 0.5]$	$[0, \frac{\pi}{2}]$
agent 2	$[-0.5, 0]$	$[0, 0.5]$	$[\frac{\pi}{2}, \pi]$
agent 3	$[-0.5, 0]$	$[-0.5, 0]$	$[\pi, \frac{3\pi}{4}]$
agent 4	$[0, 0.5]$	$[-0.5, 0]$	$[\frac{3\pi}{4}, 2\pi]$

Table 6.1: Initialization of rovers. The segments represent the range of possible initialization of the coordinate of position x, y and orientation. Each independent repetition of the algorithm uniformly samples in these ranges.

Ten POI are placed in a circle around them. The agents have T steps to reach the POI, allowing their team to get a reward. POI deliver some rewards if the n agents that visit it are in their radius of observation = 2 and respect some conditions:

- solo-POI: $r = 1$ if $n \geq 1$;
- duo-POI: $r = 3$ if $n \geq 2$.

The global return R of the team is then the sum of all rewards delivered by the POI:

$$R = \sum_{\text{POI}} r \tag{6.3}$$

Using these two types of POI, we devise two different setups, which are represented in Figure 6.3 and characterized as follows:

- **Setup 1**: 10 solo-POI are organized in a circle around the center of the arena. In this setup, to maximize the score, each agent must move to a different POI than the other agents. Since the number of POI is larger than the number of agents, this task requires a very low level of coordination between agents.
- **Setup 2**: 8 solo-POI and 2 duo-POI are organized in a circle around the center of the arena. 2 duo-POI are aligned on the horizontal axis. This setup requires a coupling of the agents' behaviour to obtain the best score. As in setup 1, the team can obtain rewards by spreading over the solo-POI. But this does not allow to get the best score (i.e. $R = 6$) as it requires that the four agents divide themselves into two teams of two to get the rewards of two duo-POI. Therefore, this task requires a higher level of coordination between agents.

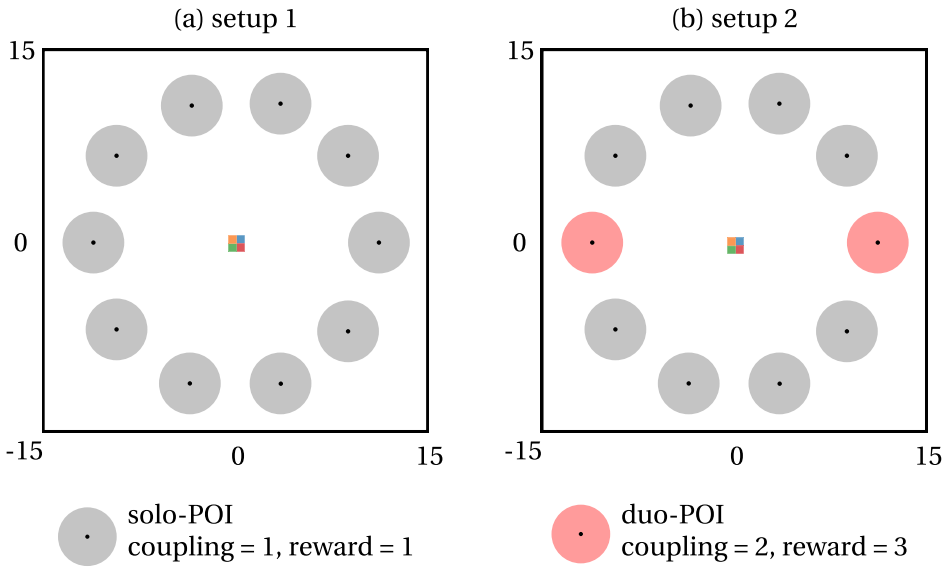


Figure 6.3: The two setups explored in this study. Setup 1 has 10 solo-POI organized in a circle around the center. We refer to this setup as the 10 solo-POI setup. Setup 2 has the same circular configuration, but the POI on the horizontal axis are duo-POI. We refer to this setup as the 8 solo 2 duo-POI setup. In both figures, the small coloured square in the middle represents the initialization area of the agents described in Table 6.1. Blue represents the initialization area of agent 1; orange represents agent 2; green represents agent 3; and red represents agent 4.

6.3.2 Robot model

Agents in the experiment represent robots subject to a kinematic model. This section describes their action space A, observation space O and the architecture of the policy π that maps the two.

Concerning the action space, it is a question of controlling a velocity vector. Thus, the agents have a 2-dimensional action space A where the two dimensions represent speed ($a_0 \in [0,1]$ for [stop, maximum speed]) and angular speed ($a_1 \in [-1,1]$ for [clockwise, anti-clockwise]).

Their observation space O is composed of :

- 3 sensors per POI that get information about their distance, angle and type;
- 2 sensors per rover that get information about their distance and angle.

Thus, for our setups with 10 POI and 4 rovers, we obtain an observation vector of $\dim(O) = 38$ dimensions.

The agent's policy π maps observations $o \in O$ to actions $a \in A$. For that purpose, we use a neural network as the main policy structure. Figure 6.4 details the full topology between inputs and outputs, and the number of parameters.

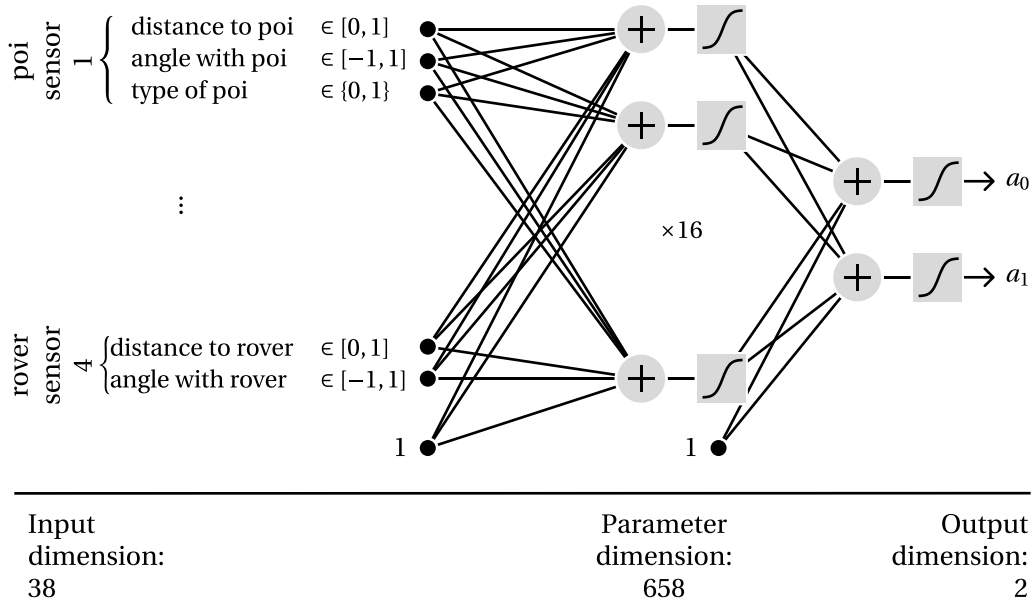


Figure 6.4: Architecture of the policy function. Each sensor gives information about the distance to obstacle and the type of object detected, if any. The architecture represented here is a multi-layered neural networks, used for the experiments. It has 38 dimensions as input, a hidden layer of 16 dimensions plus a bias term for each layer. The action dimension is 2. The total number of parameters is 658.

This architecture imposes a large number of free control parameters θ that need to be optimized. In the present case, this means $dim(\theta) = 658$ parameters.

6.4 Results

This section describes the results of applying the CC-1+1-ES algorithm to the different setups described above. The environment is stochastic, as the same policy parameters can lead to different trajectories and evaluation outcomes due to the random variation in initial conditions. Therefore, we want to estimate the fitness or expected return $F(\boldsymbol{\theta})$ of the team parameters $\boldsymbol{\theta}$.

$$F(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\tau \sim \boldsymbol{\pi}_{\boldsymbol{\theta}}} [R(\tau)] \tag{6.4}$$

Here, by construction of the task, the return of a trajectory $R(\tau)$ is the sum of the rewards given by the POIs of Equation 6.3.

During all the following experiments, we note the evaluation f to be an estimation of $F(\boldsymbol{\theta})$ by sampling the experiment multiple times on different initial conditions. More precisely, each evaluation f represents the average return of the trajectories from sixteen different initial conditions that are kept constant from one generation to the next. Using as many different repetitions for evaluation ensures that agents learn policies that generalize behaviour regardless of initial conditions.

In all experiments, we use the environment, controller and algorithm parameters detailed in Table 6.2.

Parameter	Value
POI parameters	
Number of POI	10
Radius	2
Distance from center	13
Agents parameters	
Number of agents	4
Sensor maximum distance	30px
Maximum velocity $v_{\max}$	1 px/steps
Maximum angular velocity $\omega_{\max}$	60 degrees/steps
Number of steps per episode T	15
Controller	
Initialization range	$[-2,2]$
Sensory inputs	38
Hidden layer	1
Hidden size	16
Control outputs	2
Total number of parameters	658
Algorithm parameters	
Number of repetition for evaluation f	16
Gaussian mutation probability p	0.2
Gaussian mutation size σ	0.1
Uniform mutation volume V_m	25%
Uniform mutation range $\mathscr{R}$	$[-2,2]$

Table 6.2: Common parameters of all experiments

We start by exploring with three variants of the CC-(1+1)-ES$_{k_{\text{fixed}}}$ algorithm with $k = 1, 2$ and 4, and we observe different learning dynamics depending both on the value of k

and the setup at hand. We then analyze the performances of CC-(1+1)-ES$_{k_{\text{adaptive}}}$ on the same setups.

6.4.1 Setup 1: no coupling required

We first consider setup 1 described in Figure 6.3. Here, all the points of interest give a reward of $r = 1$ if an agent is in their observation diameter at the end of the experiment. This setup is built to observe the performance of the algorithm in a task that does not require behaviour coupling. To achieve the maximum team evaluation ($R = 4$), the agents must spread out on different points of interest.

In this setup, agents are not required to coordinate to reach the same point of interest. It is not known, *a priori*, which value of k should allow the fastest convergence. Using $k = 1$ is less destructive, and agents are selected only if they individually allow a better performance. But $k > 1$ is more exploratory, and modifying several agents simultaneously can accelerate learning.

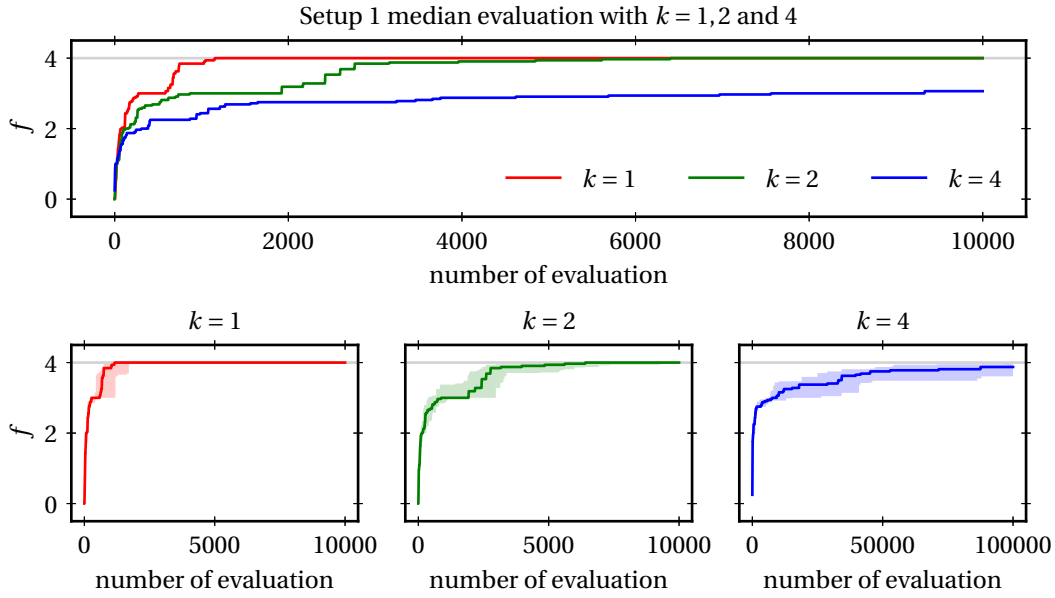


Figure 6.5: Return R for experiments with 10 solo-POI with CC-1+1-ES$_{k\text{fixed}}$, and $k = 1, 2$ and 4. On each figure, the curve represents the median on 16 independent replications, and the shaded area of the bottom figures is the interquartile range.

Figure 6.5 shows the evaluation f for group sizes of $k = 1, 2$ and 4. The three lower figures represent the medians and interquartile ranges over 16 independent experiments.

The top figure compares the medians of the different group sizes k of the algorithm.

All versions of the fixed k algorithm could reach optimal performance. When considering convergence speed, the value of $k = 1$ was found to be the fastest, followed by $k = 2$, and finally, $k = 4$, which was significantly slower. Specifically, $k = 1$ was found to be twice as fast as $k = 2$, while $k = 4$ took longer to converge, as shown in the bottom right corner of Figure 6.5. However, despite its slower convergence, $k = 4$ was still able to reach optimal performance.

The larger values of k experience degradation in performance due to concurrent exploration by multiple agents. This tends to add noise during the learning process since the evaluation variation will depend on the modification of several agents. Thus, bad policies for one agent can be selected as long as the overall modification is favourable. And conversely, good modifications could be missed if the teammates didn't follow up. These are similar to the inter-agent exploration noise highlighted by Chung et al., 2018; HolmesParker et al., 2014. Furthermore, in this setup, no behavioural coupling is required, and the agents can essentially act independently.

6.4.2 Setup 2: joint strategies

Setup 2, on the other hand, has been created to require the coupling of behaviours between agents to achieve the best evaluation. As in setup 1, agents can get rewards by splitting up into solo-POIs. But this will not give the best score obtainable by the team. To get it, they have to coordinate and go to the two duo-POIs in groups of two. The objective of this setup is, therefore, to observe how the need for behaviour coupling will impact the performance of the algorithm.

Figure 6.6 shows the evaluation f for group sizes of $k = 1, 2$ and 4. The three lower figures represent the medians and interquartile ranges over 16 independent experiments. The top figure allows for a comparison of the medians of the different group sizes k of the algorithm.

In setup 2, we observe that the three versions of the algorithm CC-1+1-ES$_{k_{\text{fixed}}}$ quickly reach an evaluation threshold. This evaluation corresponds to a situation where the 4 agents are on solo-POI, as in setup 1. We find the same performances as before, that is to say, $k = 1$ allows faster improvement than $k = 2$, itself faster than $k = 4$.

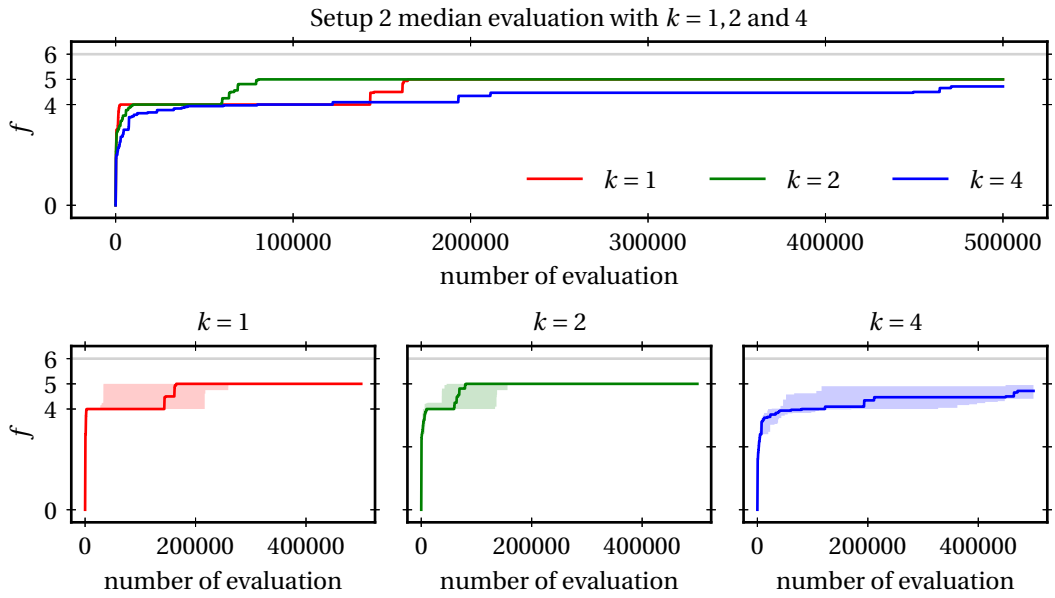


Figure 6.6: Return R for experiments with 8 solo-POI and 2 duo-POI, with CC-1+1-ES$_{k\text{fixed}}$, and $k = 1, 2$ and 4. On each figure, the curve represents the median on 16 independent replications, and the shaded area of the bottom figures is the interquartile range.

To increase evaluation, agents need to move in pairs on duo-POIs. Figure 6.7 represents the learning curves of sixteen independent runs for $k = 1, 2$ and 4. The bold vertical dotted lines represent the number of generations from which half of the runs have reached an evaluation of 5. This time, using $k = 2$ increases the score most efficiently, and half of the runs reach a score of 5 twice as fast as using $k = 1$. These evaluation thresholds are not yet reached with $k = 4$. Only a few runs get the optimal value where agents reach the two duo-POI. But since it happens, we can make the assumption that with an arbitrarily large number of generations, all runs will eventually converge to the optimal value.

This superiority of $k = 2$ can be explained by the necessity of a joint modification in order to increase the score. As two agents must simultaneously move from their individual solo-POI to a shared duo-POI, the probability of this happening is greater when the behaviour of several agents is directly modified simultaneously. But this also means that it is surprising that the method with $k = 1$ reaches scores requiring the coupling of agents.

Moreover, Figure 6.7 allows us to see that some runs obtain the optimal evaluation

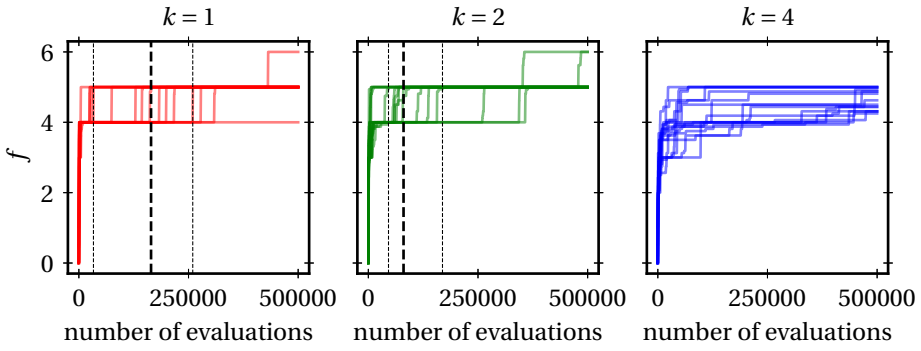


Figure 6.7: Learning curves of the sixteen independent replication of the method for $k = 1, 2$ and 4. The vertical dotted lines represent the number of evaluations required for respectively 1/4 (thin), 1/2 (bold) and 3/4 (thin) of the run to reach a score $f \geq 5$. Half of the experiments reach a score of $f = 5$ in 80264 evaluations using $k = 2$, and 164580 evaluations are required for the same outcome using $k = 1$.

using $k = 2$ and $k = 1$. The next section shows how the modification of one agent allows the change of trajectory of several and, thus, that the learning of coordinated behaviour is possible even using $k = 1$.

6.4.3 Behavioural transition

Modifying a single agent at each generation ($k = 1$) still allows for joint trajectory modification. This can be a surprise because changing the parameters of an agent should only change its behaviour. But as the agents can perceive each other, they react to the trajectory of their teammates. There is a coupling of behaviours by construction of the sensory inputs. Thus, the value of k does not necessarily prevent the simultaneous modification of multiple agent trajectories.

In this section, we analyze the dependency between the agents' behaviours to better understand how a coordinated behaviour can emerge through the sequential modification of a single agent at a time.

Figure 6.8 shows examples of trajectory on a run using $k = 1$ at two specific time of learning. On the left are represented the trajectory of the agents having found a solution to obtain a score of $f = 5$. On the right are represented the trajectory of the agents after obtaining the optimal score $f = 6$.

To better understand agents' dependence on each other, we can immobilize their teammates and then compare their trajectories with and without the other agents. The middle graphs of Figure 6.8 represent the trajectories of each agent when their teammates are inactive. Modification of trajectories is often minimal and just end-of-path adjustments, which can be justified by their observation dependence on each other. Most agents have independent behaviours.

However, the case of agent 2 is interesting. Its trajectory has largely deviated in the absence of its teammates. In particular, after obtaining the maximum score (on the right), agent 2 has trajectories that stay in the center of the zone, where the other agents are immobilized. When they are all active (top right figure), agent 2 sees his trajectory unfold to follow agent 3 on the left POI. Its follower behaviour allowed the team to increase its score by modifying only one agent at a time.

6.4.4 Adaptive methods experiments

In this section, we compare the performance of CC-1+1-ES$_{k_{\text{adaptive}}}$, which automatically adjusts the group size k at each generation using the EXP-3 algorithm.

Figure 6.9 shows the median on sixteen independent runs of the evaluation f of the adaptive methods.

On setup 1, while adaptive methods are able to approach the performance of $k = 1$ in certain cases, they can't fully achieve their benefits due to the exploration cost associated with varying the group size k. It still reaches the best possible score faster than $k = 2$ and $k = 4$. Similarly, on setup 2, adaptive methods allow for closely following the best possible k choice.

Using the adaptive methods allows achieving results similar to the best choice of k, but doesn't require any *a priori* on its value.

6.5 Conclusion

In this chapter, we adapted the CC-1+1-ES algorithms to a classic multi-robot setup. In these tasks, agents must distribute themselves on points of interest distributed around them. Some of these points of interest require agent coordination to deliver rewards.

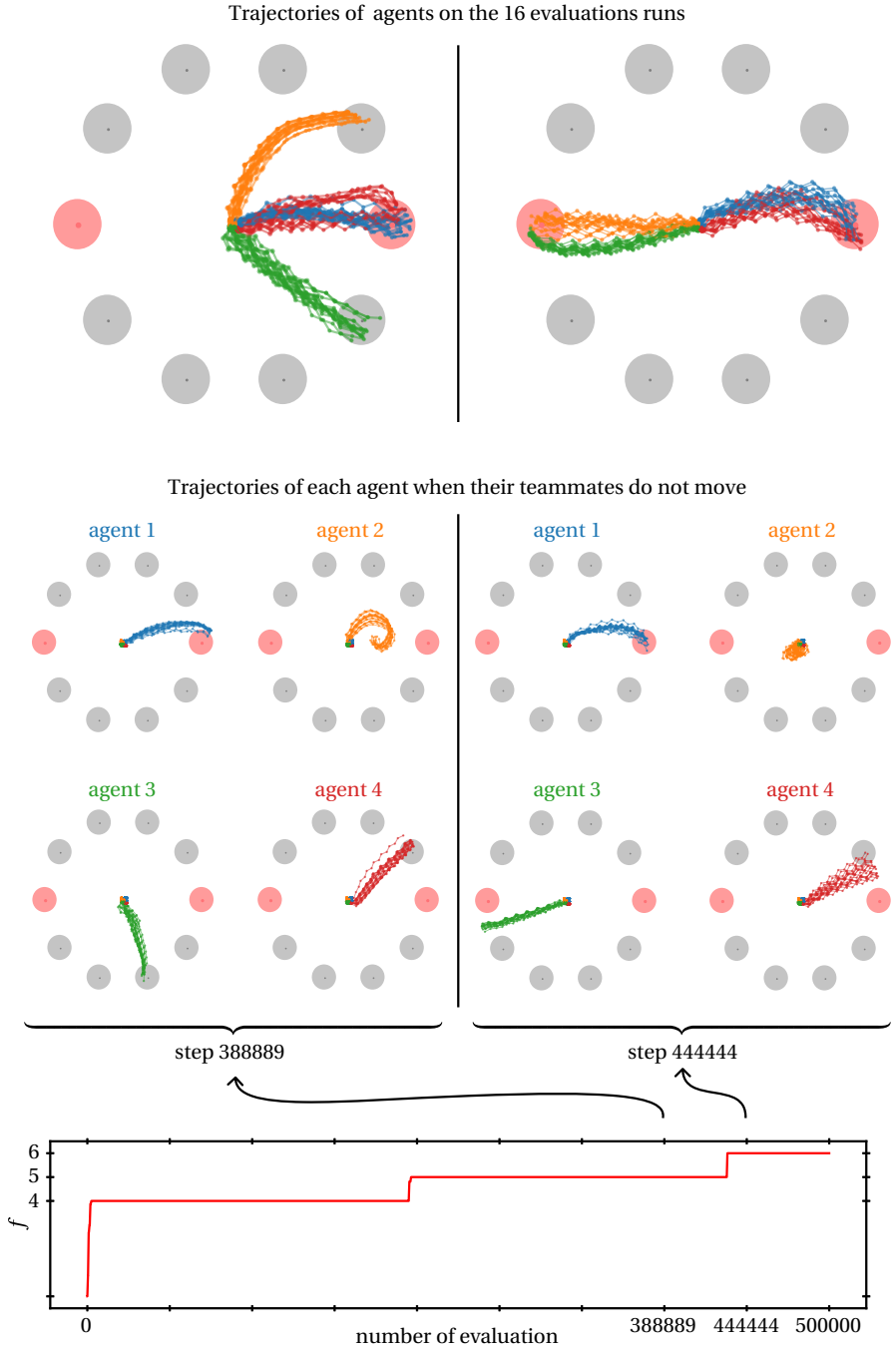


Figure 6.8: Trajectories of agents at evaluation 388889 and 444444 for a selected $k = 1$ run. The different traces represent the trajectories generated from the sixteen initial conditions used for evaluation of the policies. The top-row figures shows the trajectories of the fours agent acting simultaneously. The middle-row shows the trajectories of each agent when their teammate actions are nullified. The bottom plot is score evolution of the run whose parameters are extracted.

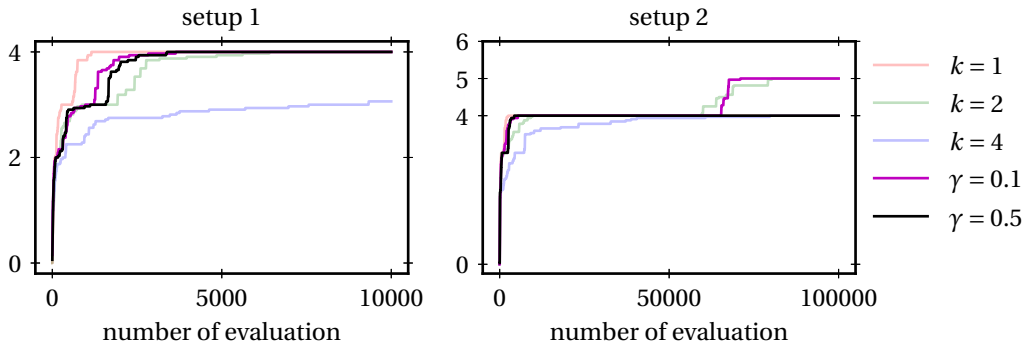


Figure 6.9: Median of return R on 16 independent replications for setups with 10 solo-POI (left) and 8 solo-POI and 2 duo-POI (right) using CC-1+1-ES$_{k\text{adaptive}}$ with various γ. Lower opacity of the CC-1+1-ES$_{k\text{fixed}}$ with $k = 1, 2$ and 4 are plotted as references.

This task is, therefore, an example of an environment where it is useful to jointly modify agents, allowing them to discover coordination behaviours. However, joint modification causes noise in the evaluation of the agents, who have to distinguish between their individual contributions and their teammates.

The results presented in this chapter showed that the number of agents modified at each generation by CC-1+1-ES, allowing the best performances, is well related to the degree of coordination of the task. Since the agents can observe each other, behavioural coupling can emerge even by sequentially changing only one agent at a time. However, we still observe that a task requiring the coupling of behaviours largely benefits from the joint modification of the policies of several agents.

Finally, without any *a priori* knowledge of the degree of coordination required, it is still possible to adapt the number of modified agents automatically using the EXP-3 algorithm and obtain good performances.

7 Conclusion

7.1 Summary

Chapter 2 discussed the different types of multi-robot systems, where robots are designed to work together to perform tasks. They differ in the way robots interact with each other through communication networks, a central authority and a possible human control. These architectures will depend on the applications, physical, technological, and cost constraints. We have highlighted some of the current applications of multi-robot systems such as warehouse management, surveillance, agriculture or drone shows. These applications require cooperation and collective decision behaviours that researchers and engineers are trying to implement. We can think in particular of collective decision-making, consensus, task allocation, or collective movement and positioning. Multiple methods have been used over the years to implement those. Most current applications rely on manual programming as they offer more predictable behaviour and are easier to deploy on many robots. But the need for adaptability to a changing environment and the intuition for design behaviour can be challenging. Therefore, pushed by significant advances in machine learning algorithms, many new methods have been proposed to optimize multi-robots behaviours.

Chapter 3 presented a framework for the optimization of multiple robots, which are described as agents interacting with an environment. The robots use a policy function that must be optimized, but there are difficulties in handling the reward used for optimization in the multi-agent setting. One notable difficulty is the social welfare versus credit assignment dilemma, where there may be a conflict between the

accuracy of an individual learning signal of the agents and the system's overall goals. The possible trade-offs in this dilemma are limited by the possible information sharing between agents. If no information transit between agents, the possible mitigation is limited, but if agents are able to share information through a communication network of a central authority, more techniques of reward marginalization or decomposition can be deployed. Therefore, the reward and information exchange architecture both impact the policy optimization algorithm. Model-free methods can be decomposed into classes, including reinforcement learning algorithms and direct policy search with evolutionary algorithms. This thesis focused on the latter as they allow more lightweight methods relevant to robotic learning. Using this framework, we studied two very different problems on the collective robotic learning spectrum.

In **Chapter 4**, we studied the dynamics of a team of networked learning agents using embodied evolution algorithms. Agents are deployed in their environment, where they can interact with each other and share information through a temporal network that depends on agents' communication radius. In order to improve their performance, we introduce an algorithm called Horizontal Information Transfer. This algorithm allows agents to estimate their own performance on the fly and share policy parameters using a transfer operator. We tested this algorithm on a simulated foraging task and showed that it allows the agents to reach good performance levels. However, there is a trade-off between the speed at which the agents can learn and the accuracy of their performance. This trade-off is similar to the exploration vs exploitation trade-off that is commonly encountered in machine learning. Furthermore, the meta parameters of the algorithm can be learned on the fly, allowing the agents to adapt to their environment and improve their performance even further.

Chapter 5 tackled another coordination problem. We study the situation where agents are able to interact with their environment through episodes and can share information and synchronize between these interactions. Importantly, the agents are rewarded at the team level rather than at the individual level. To address this optimization problem, we introduce a cooperative co-evolutionary algorithm called CC-1+1-GA. This algorithm uses an evolutionary optimization procedure in a framework that resembles a hill-climbing algorithm. It gradually improves individual robots by estimating the collective gains provided by new individual behaviours. The number of agents modified at each generation is shown to influence the learning dynamics, and we propose a method for computing an optimal group size online using the EXP3

algorithm. We then test our CC-1+1-GA algorithm on a variation of the El-Farol Bar problem, which highlights the differences in learning dynamics and the advantages of using different group sizes. Overall, the results of this chapter show the effectiveness of the CC-1+1-GA algorithm in addressing the problem of coordination. It is possible to provide efficient anytime learning by modulating the number of modified agents over time using the EXP-3 algorithm. This approach does not require any prior knowledge about the problem to be solved, and can be effective in situations where coupling between agents during learning is necessary to improve team performance.

Moreover, we detail the application of the latter algorithm to a more realistic robotic rover task. In these experiments, the agents must physically distribute themselves on points of interest with a control policy. Obtaining a reward is thus not only conditioned by the coordination between agents but also by their own behaviour and trajectories. It is difficult to distinguish whether a score variation comes from one's own behaviour or from the exploration of other agents. It is, therefore, critical to proceed cautiously by modifying only one agent at each generation. But this is problematic when the problem requires coupling of the agents' behaviour. In our rover experiment, for example, some points of interest require the coordination of two agents to reward the team. It is, therefore, necessary to modify several agents simultaneously. **Chapter 6** shows hows the CC-1+1-ES extends Chapter 5 algorithms and handles these issues of agent exploration noise and coupling. It shows how the best group size parameter reflects the amount of coupling required to solve a task. And the adaptive version of the algorithms allows for easy group size fine-tuning.

7.2 Discussion and perspectives

The results described in this manuscript could lead to many further studies, some of which are described hereafter.

An obvious limitation of these studies concerns the use of robotic simulations. While our findings are primarily theoretical and are not tied to a specific robot model, we are aware of the open issue of transferability to real-world robots (i.e. the reality gap). Implementation on real robots (Kilobots) of the HIT algorithm has been done by members of the team and confirms the feasibility of the method to optimize robots after deployment (Mirhosseini et al., 2022; Zion et al., 2021). Implementations on real

robots of the CC-1+1-GA algorithms remain to be done.

7.2.1 Perspective on social learning for swarm robotics

Our experiments on lifetime learning focused on an object foraging task where objects were uniformly distributed in the environment. To solve the task, each individual had to spread out in the arena, which allowed them to better transmit their parameters. This created a positive feedback loop that reinforced the best parameters in the population. However, what happens when the task does not encourage parameter transmission, or when the robot density is low? The deterministic loss of messages may also be questioned. How robust is it to a transfer rate which would be noisy, i.e. for each message, the likelihood of the message being lost is drawn, instead of assuming that all messages larger than a certain threshold are lost?

These questions have been explored in a recent study, as detailed in Mirhosseini et al., 2022. For example, It may be very interesting to explore the parallel use of individual learning to compensate for the possible lack of encounter with other agents.

The credit assignment of agents could also be questioned. In our foraging experiment, individual assessment is performed using a sliding window on the last obtained rewards. This allowed an adaptation of all agents in case the environment is changing. However, this assessment is very noisy, and performing policies can eventually be discarded due to bad luck. More precise rewarding schemes can be achieved using peer evaluation and asking neighbours about one's potential impact on their assessment. For example, when two agents compete for an object, they could exchange penalties for agents that would have taken an object in front of them or share its benefits.

Finally, the method we propose allows for finding consensus on control parameters that work well for the swarm. But sometimes, such consensus is not desirable, and we may want to derive specialization among agents. The solution to this issue lies in the transfer operator and its selection mechanism. That is to say, *from which* agents and *how* shared parameters can be integrated into the current active parameters. For example, a simple heterogeneous team could be achieved through a clear type definition, where agents can share parameters only if they have the same type. However, this would require an *a priori* knowledge on the required distribution of each type.

7.2.2 Perspective on cooperative co-evolution for multi-robot systems

In Chapter 6, our experiments have only been done on systems that are relatively far from real robotics, especially in terms of their observations of the world. In particular, the rover experiment admits the use of distance and angle sensors with objects rarely accessible via sensors in the real world. Robots generally use cameras or lidar to locate themselves in space. Other experiments should be conducted to estimate the performance of our methods on more robotic systems.

Concerning the co-evolutionary algorithm, our methods use a simple hill-climbing algorithm for each individual. It allows for reducing the possible context agents and helps the learning stability. However, it drastically reduces the exploration capabilities of the algorithm. More complex evolutionary algorithms could be used, such as Natural Evolution Strategies (NES). Instead of relying on a sample population of parameters, this class of algorithm maintain a probability distribution of parameters and sample agent from it.

Figure 7.1 is a representation of such algorithms. The overall evaluation depends on all agents. In the figure, the sampling of the parameters presented is independent of the teammates. But future research could also integrate into these algorithms an evaluation of the correlation between the different agent parameters in order to adapt the contextual agents to the exploration of an agent.

7.2.3 Merging the two problems

One could also consider merging the two problems tackled in this thesis using a single online optimization method. For example, consider a homogeneous team of agents that have evolved to solve a task. To test new strategies, we introduce mutations in the population. Using a very low mutation rate, we can hope that only one agent in the swarm is different and thus measure more precisely the effect of the change on its behaviour and neighbours.

Then a beneficial mutation will tend to propagate well in the population, allowing agents to obtain better rewards. The evaluation of the quality of a mutation is also linked to its ability to propagate in the swarm. Several agents can introduce mutations

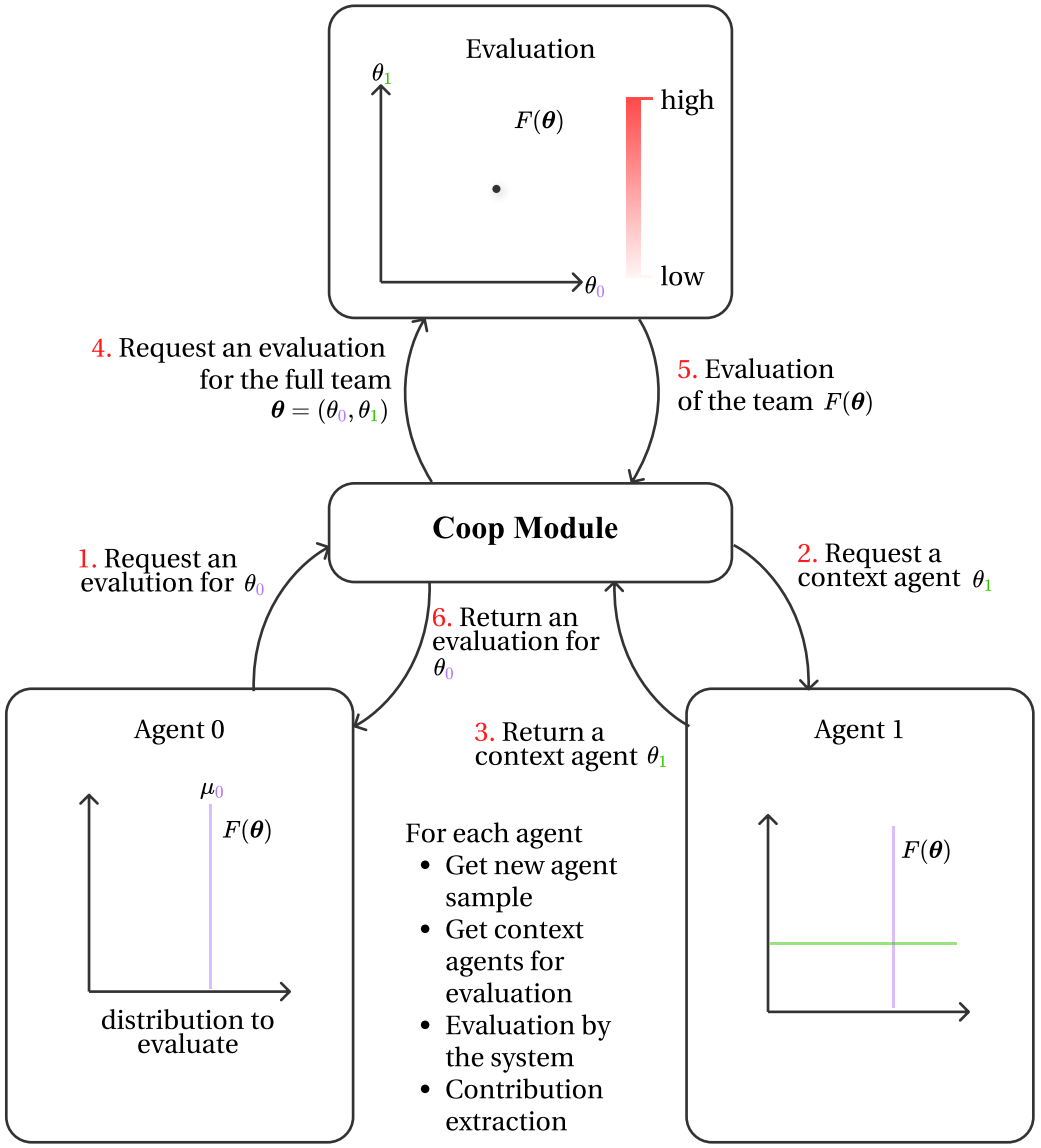


Figure 7.1: Diagram of a general CCEA algorithm centred around message exchange between agents and a cooperation module that interfaces with the evaluation environment. This highlights the generalization of the algorithm to individual methods centred on NES

to avoid waiting for the propagation or complete disappearance of a mutation. But, this will obstruct their evaluation by the swarm because the mutants will compete with several other agent policies.

We thus find a dilemma similar to the number of mutated agents between each generation of our algorithms CC-1+1-GA and CC-1+1-ES. These could allow for more reliable increases in the swarm's performances. Analyses could also allow us to understand better the dynamics of the link between evaluation and propagation in temporal network algorithms.

Bibliography

Ackerman, E. (2011). *Matternet wants to deliver meds with a network of quadrotors*. Retrieved July 16, 2022, from https://spectrum.ieee.org/mini-uavs-could-be-the-cheapest-way-to-deliver-medicine

Agogino, A. K., HolmesParker, C., & Tumer, K. (2012a). Evolving distributed resource sharing for cubesat constellations. *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, 1015–1022.

Agogino, A. K., HolmesParker, C., & Tumer, K. (2012b). Evolving large scale uav communication system. *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, 1023–1030.

Agogino, A. K., & Tumer, K. (2004a). Efficient evaluation functions for multi-rover systems. In K. Deb (Ed.), *Genetic and evolutionary computation – gecco 2004* (pp. 1–11). Springer Berlin Heidelberg.

Agogino, A. K., & Tumer, K. (2004b). Efficient evaluation functions for multi-rover systems. *Genetic and evolutionary computation conference*, 1–11.

Agogino, A. K., & Tumer, K. (2008). Analyzing and visualizing multiagent rewards in dynamic and stochastic domains. *Autonomous Agents and Multi-Agent Systems*, *17*(2), 320–338.

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002). Wireless sensor networks: a survey. *Computer networks*, *38*(4), 393–422.

Albus, J. S. (1991). Outline for a theory of intelligence. *IEEE transactions on systems, man, and cybernetics*, *21*(3), 473–509.

Alhafnawi, M., Hauert, S., & O'Dowd, P. (2020). Robotic canvas: interactive painting onto robot swarms. *ALIFE 2020: The 2020 Conference on Artificial Life*, 163–170.

Alonso-Mora, J., Schoch, M., Breitenmoser, A., Siegwart, R., & Beardsley, P. (2012). Object and animation display with multiple aerial vehicles. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1078–1083.

Alsamhi, S. H., & Lee, B. (2020). Blockchain-empowered multi-robot collaboration to fight covid-19 and future pandemics. *Ieee Access*, *9*, 44173–44197.

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., & Zaremba, W. (2017). Hindsight experience replay. *Advances in neural information processing systems*, *30*.

Aragues, R., Cortes, J., & Sagues, C. (2012). Distributed consensus on robot networks for dynamically merging feature-based maps. *IEEE Transactions on Robotics*, *28*(4), 840–854.

Arkin, R. C. et al. (1998). *Behavior-based robotics*. MIT press.

Arthur, W. B. (1994). Inductive reasoning and bounded rationality. *The American Economic Review*, *84*(2), 406–411. http://www.jstor.org/stable/2117868

Asaro, P. (2012). On banning autonomous weapon systems: human rights, automation, and the dehumanization of lethal decision-making. *International review of the Red Cross*, *94*(886), 687–709.

Auer, P., Cesa-Bianchi, N., Freund, Y., & Schapire, R. E. (2002). The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, *32*(1), 48–77. https://doi.org/10.1137/S0097539701398375

Augugliaro, F., Lupashin, S., Hamer, M., Male, C., Hehn, M., Mueller, M. W., Willmann, J. S., Gramazio, F., Kohler, M., & D'Andrea, R. (2014). The flight assembled architecture installation: cooperative construction with flying machines. *IEEE Control Systems Magazine*, *34*(4), 46–64.

Baldassarre, G., Nolfi, S., & Parisi, D. (2003). Evolving mobile robots able to display collective behaviors. *Artificial life*, *9*(3), 255–267.

Baray, C. (1997). Evolving cooperation via communication in homogeneous multi-agent systems. *Proceedings Intelligent Information Systems. IIS'97*, 204–208.

Barber, R., Ortiz, F. J., Garrido, S., Calatrava-Nicolás, F. M., Mora, A., Prados, A., Vera-Repullo, J. A., Roca-González, J., Méndez, I., & Mozos, Ó. M. (2022). A multi-robot system in an assisted home environment to support the elderly in their daily lives. *Sensors*, *22*(20), 7983.

Bautin, A., Simonin, O., & Charpillet, F. (2012). Minpos: a novel frontier allocation algorithm for multi-robot exploration. *International conference on intelligent robotics and applications*, 496–508.

Bayindir, L. (2016). A review of swarm robotics tasks. *Neurocomputing*, *172*, 292–321. https://doi.org/10.1016/j.neucom.2015.05.116

Bechar, A., & Vigneault, C. (2016). Agricultural robots for field operations: concepts and components. *Biosystems Engineering*, *149*, 94–111.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., & Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, *29*.

Benarbia, T., & Kyamakya, K. (2021). A literature review of drone-based package delivery logistics systems and their implementation feasibility. *Sustainability*, *14*(1), 360.

Beni, G. (2004). From swarm intelligence to swarm robotics. *International Workshop on Swarm Robotics*, 1–9.

Beni, G. (2005). From Swarm Intelligence to Swarm Robotics. *Robotics*, *3342*, 1–9. https://doi.org/10.1007/978-3-540-30552-1{_}1

Bernard, A., André, J.-B., & Bredeche, N. (2016). To cooperate or not to cooperate: why behavioural mechanisms matter. *PLoS computational biology*, *12*(5), e1004886.

Bernstein, D. S., Givan, R., Immerman, N., & Zilberstein, S. (2002). The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, *27*(4), 819–840.

Beyer, H.-G., & Schwefel, H.-P. (2002). Evolution strategies - a comprehensive introduction. *Natural Computing*, *1*(1), 3–52.

Beynier, A., Charpillet, F., Szer, D., & Mouaddib, A.-I. (2013). Dec-mdp/pomdp. *Markov Decision Processes in Artificial Intelligence*, 277–318.

Bloembergen, D., Tuyls, K., Hennes, D., & Kaisers, M. (2015). Evolutionary dynamics of multi-agent learning: a survey. *Journal of Artificial Intelligence Research*, *53*, 659–697.

Bonabeau, E., Dorigo, M., Theraulaz, G., & Theraulaz, G. (1999). *Swarm intelligence: from natural to artificial systems*. Oxford university press.

Bongard, J. C., & Paul, C. (2000). Investigating morphological symmetry and locomotive efficiency using virtual embodied evolution. *From Animals to Animats: The Sixth International Conference on the Simulation of Adaptive Behaviour*.

Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. *TARK*, *96*, 195–210.

Bowling, M. H., Browning, B., & Veloso, M. M. (2004). Plays as effective multiagent plans enabling opponent-adaptive play selection. *ICAPS*, 376–383.

Boysen, N., De Koster, R., & Weidinger, F. (2019). Warehousing in the e-commerce era: a survey. *European Journal of Operational Research*, *277*(2), 396–411.

Braitenberg, V. (1984). Vehicles.

Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2012). Swarm robotics : A review from the swarm engineering perspective. *Swarm Intelligence*, *7*(1), 1–41. https://doi.org/10.1007/s11721-012-0075-2

Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2013). Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, *7*(1), 1–41. https://doi.org/10.1007/s11721-012-0075-2

Brambilla, M., Pinciroli, C., Birattari, M., & Dorigo, M. (2012). Property-driven design for swarm robotics. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, 139–146.

Bredeche, N., Haasdijk, E., & Prieto, A. (2018). Embodied evolution in collective robotics: a review. *Frontiers in Robotics and AI*, *5*, 12. https://doi.org/10.3389/frobt.2018.00012

Bredeche, N., & Montanier, J.-m. (2010). Environment-driven Embodied Evolution in a Population of Autonomous Agents. *Parallel Problem Solving from Nature (PPSN)*, 290–299.

Bredeche, N., Montanier, J.-M., Liu, W., & Winfield, A. F. T. (2012). Environment-driven Distributed Evolutionary Adaptation in a Population of Autonomous Robotic Agents. *Mathematical and Computer Modelling of Dynamical Systems*, *18*(1), 101–129. https://doi.org/10.1080/1387395YYxxxxxxxx

Bredeche, N., Montanier, J., Weel, B., & Haasdijk, E. (2013). Roborobo! a fast robot simulator for swarm and collective robotics. *CoRR*, *abs/1304.2888*. http://arxiv.org/abs/1304.2888

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, *2*(1), 14–23.

Brown, G. W. (1951). Iterative solution of games by fictitious play. *Act. Anal. Prod Allocation*, *13*(1), 374.

Brown, N., & Sandholm, T. (2019). Superhuman ai for multiplayer poker. *Science*, *365*(6456), 885–890.

Bryant, B. D., & Miikkulainen, R. (2003). Neuroevolution for adaptive teams. *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, *3*, 2194–2201.

Campbell, M., Hoane Jr, A. J., & Hsu, F.-h. (2002). Deep blue. *Artificial intelligence*, *134*(1-2), 57–83.

Castelló Ferrer, E. (2018). The blockchain: a new framework for robotic swarm systems. *Proceedings of the future technologies conference*, 1037–1058.

Cheein, F. A. A., & Carelli, R. (2013). Agricultural robotics: unmanned robotic service units in agricultural tasks. *IEEE industrial electronics magazine*, *7*(3), 48–58.

Chen, J., Song, L., Wainwright, M. J., & Jordan, M. I. (2018). L-shapley and c-shapley: efficient model interpretation for structured data. *arXiv preprint arXiv:1808.02610*.

Chen, J., Yang, Y., & Wei, L. (2010). Research on the approach of task decomposition in soccer robot system. *2010 International Conference on Digital Manufacturing & Automation*, *2*, 284–289.

Choi, H.-L., Brunet, L., & How, J. P. (2009). Consensus-based decentralized auctions for robust task allocation. *IEEE transactions on robotics*, *25*(4), 912–926.

Chung, J. J., Chow, S., & Tumer, K. (2018). When less is more: reducing agent noise with probabilistically learning agents. *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 1900–1902.

Claus, C., & Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, 746–752.

Cliff, O. M., Fitch, R., Sukkarieh, S., Saunders, D. L., & Heinsohn, R. (2015). Online localization of radio-tagged wildlife with an autonomous aerial robot system. *Robotics: Science and Systems*.

Colby, M., Duchow-Pressley, T., Chung, J. J., & Tumer, K. (2016). Local approximation of difference evaluation functions. *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, 521–529.

Costelha, H., & Lima, P. (2012). Robot task plan representation by petri nets: modelling, identification, analysis and execution. *Autonomous Robots*, *33*(4), 337–360.

Cruciol, L. L., de Arruda Jr, A. C., Weigang, L., Li, L., & Crespo, A. M. (2013). Reward functions for learning to control in air traffic flow management. *Transportation Research Part C: Emerging Technologies*, *35*, 141–155.

Cubber, G. D., Doroftei, D., Rudin, K., Berns, K., Serrano, D., Sanchez, J., Govindaraj, S., Bedkowski, J., & Roda, R. (2017). Search and rescue robotics-from theory to practice.

Cummings, M. L. (2004). Human supervisory control of swarming networks. *2nd annual swarming: autonomous intelligent networked systems conference*, 1–9.

De Greeff, J., Mioch, T., Van Vught, W., Hindriks, K., Neerincx, M. A., & Kruijff-Korbayová, I. (2018). Persistent robot-assisted disaster response. *Companion of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, 99–100.

De Jong, K. A. (2016). Evolutionary computation: a unified approach. *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, 185–199.

Dietz, G., E, J. L., Washington, P., Kim, L. H., & Follmer, S. (2017). Human perception of swarm robot motion. *Proceedings of the 2017 CHI conference extended abstracts on human factors in computing systems*, 2520–2527.

Dixit, G., & Tumer, K. (2022). Behavior exploration and team balancing for heterogeneous multiagent coordination. *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 1578–1579.

Dixit, G., Zerbel, N., & Tumer, K. (2019). Dirichlet-multinomial counterfactual rewards for heterogeneous multiagent systems. *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, 209–215.

Douchan, Y., Wolf, R., & Kaminka, G. A. (2019). Swarms can be rational. *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 149–157.

Dow, J., Matussi, S., Dow, R. M., Schmidt, M., & Warhaut, M. (2004). The implementation of the cluster ii constellation. *Acta Astronautica*, *54*(9), 657–669.

Duarte, M., Gomes, J., Costa, V., Rodrigues, T., Silva, F., Lobo, V., Marques, M. M., Oliveira, S. M., & Christensen, A. L. (2016). Application of swarm robotics systems to marine environmental monitoring. *OCEANS 2016-Shanghai*, 1–8.

Dutta, A., Roy, S., Kreidl, O. P., & Bölöni, L. (2021). Multi-robot information gathering for precision agriculture: current state, scope, and challenges. *IEEE Access*, *9*, 161416–161430. https://doi.org/10.1109/ACCESS.2021.3130900

Ecoffet, P., Fontbonne, N., André, J.-B., & Bredeche, N. (2022). Policy search with rare significant events: choosing the right partner to cooperate with. *PloS one*, *17*(4), e0266841.

Eiben, A. E., & Smith, J. E. (2003). *Introduction to evolutionary computing* (Vol. 53). Springer.

Eiben, A. E., Smith, J. E. et al. (2003). *Introduction to evolutionary computing* (Vol. 53). Springer.

Enright, J. J., & Wurman, P. R. (2011). Optimization and coordinated autonomy in mobile fulfillment systems. *Workshops at the twenty-fifth AAAI conference on artificial intelligence*.

Escoubet, C. P. (2000). Cluster-ii: scientific objectives and data dissemination. *ESA bulletin*, 54–60.

Escoubet, C., Schmidt, R., & Goldstein, M. (1997). Cluster-science and mission overview. *The cluster and phoenix missions*, 11–32.

Evans, W. (2020). *How amazon hid its safety crisis*. Retrieved July 16, 2022, from https://revealnews.org/article/how-amazon-hid-its-safety-crisis/

Farinelli, A., Iocchi, L., & Nardi, D. (2004). Multirobot systems: a classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *34*(5), 2015–2028.

Fernandez Pérez, I., Boumaza, A., & Charpillet, F. (2014). Comparison of selection methods in on-line distributed evolutionary robotics. *Proceedings of the fourteenth international conference on the synthesis and simulation of living systems*, 1–16.

Ferrante, E., Turgut, A. E., Duéñez-Guzman, E., Dorigo, M., & Wenseleers, T. (2015). Evolution of Self-Organized Task Specialization in Robot Swarms. *PLoS Computational Biology*, *11*(8), e1004273. https://doi.org/10.1371/journal.pcbi.1004273

Ferri, G., Jakuba, M. V., Mondini, A., Mattoli, V., Mazzolai, B., Yoerger, D. R., & Dario, P. (2011). Mapping multiple gas/odor sources in an uncontrolled indoor environment using a bayesian occupancy grid mapping based method. *Robotics and Autonomous Systems*, *59*(11), 988–1000.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., & Whiteson, S. (2017). Counterfactual multi-agent policy gradients.

Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P. H., Kohli, P., & Whiteson, S. (2017). Stabilising experience replay for deep multi-agent reinforcement learning. *International conference on machine learning*, 1146–1155.

Frye, C., Rowat, C., & Feige, I. (2020). Asymmetric shapley values: incorporating causal knowledge into model-agnostic explainability. *Advances in Neural Information Processing Systems*, *33*, 1229–1239.

Fulda, N., & Ventura, D. (2007). Predicting and preventing coordination problems in cooperative q-learning systems. *IJCAI*, *2007*, 780–785.

Funes, P., & Pujals, E. (2005). Intransitivity revisited coevolutionary dynamics of numbers games. *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, 515–521. https://doi.org/10.1145/1068009.1068095

García-Pérez, L., García-Alegre, M., Ribeiro, A., & Guinea, D. (2008). An agent of behaviour architecture for unmanned control of a farming vehicle. *computers and electronics in agriculture*, *60*(1), 39–48.

Garnier, S., Jost, C., Jeanson, R., Gautrais, J., Asadpour, M., Caprari, G., & Theraulaz, G. (2005). Aggregation behaviour as a source of collective decision in a group of cockroach-like-robots. *European conference on artificial life*, 169–178.

Gerkey, B. P., & Matarić, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *The International journal of robotics research*, *23*(9), 939–954.

Gintis, H. et al. (2000). *Game theory evolving: a problem-centered introduction to modeling strategic behavior*. Princeton university press.

Glad, A., Simonin, O., Buffet, O., & Charpillet, F. (2008). Theoretical study of ant-based algorithms for multi-agent patrolling. *Ecai 2008* (pp. 626–630). IOS press.

Gmytrasiewicz, P. J., & Doshi, P. (2004). Interactive pomdps: properties and preliminary results. *International Conference on Autonomous Agents: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-*, *3*, 1374–1375.

Gomes, J., Mariano, P., & Christensen, A. L. (2017). Novelty-driven cooperative coevolution. *Evolutionary computation*, *25*(2), 275–307.

Gomes, J., Mariano, P., & Christensen, A. L. (2018). Dynamic team heterogeneity in cooperative coevolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, *22*(6), 934–948. https://doi.org/10.1109/TEVC.2017.2779840

Gomes, J., Mariano, P., & Christensen, A. L. (2019). Challenges in cooperative coevolution of physically heterogeneous robot teams. *Natural Computing*, *18*(1), 29–46. https://doi.org/10.1007/s11047-016-9582-1

Gottipati, S. K., Pathak, Y., Nuttall, R., Chunduru, R., Touati, A., Subramanian, S. G., Taylor, M. E., Chandar, S., et al. (2020). Maximum reward formulation in reinforcement learning. *arXiv preprint arXiv:2010.03744*.

Grasse, P.-P. (1959). La reconstruction du nid et les coordinations interindividuelles chezbellicositermes natalensis etcubitermes sp. la théorie de la stigmergie: essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, *6*, 41–80.

Grayson, S. (2014). Search & rescue using multi-robot systems. *School of Computer Science and Informatics, University College Dublin*, 231–432.

Groβ, R., & Dorigo, M. (2008). Evolution of solitary and group transport behaviors for autonomous robots capable of self-assembling. *Adaptive Behavior*, *16*(5), 285–305.

Grogan, S., Pellerin, R., & Gamache, M. (2018). The use of unmanned aerial vehicles and drones in search and rescue operations–a survey. *Proceedings of the PROLOG*.

Groß, R., & Dorigo, M. (2008). Self-assembly at the macroscopic scale. *Proceedings of the IEEE*, *96*(9), 1490–1508.

Gubrud, M. (2014). Stopping killer robots. *Bulletin of the Atomic Scientists*, *70*(1), 32–42.

Guerrero-Bonilla, L., Prorok, A., & Kumar, V. (2017). Formations for Resilient Robot Teams. *IEEE Robotics and Automation Letters*, *2*(2), 841–848. https://doi.org/10.1109/LRA.2017.2654550

Hamann, H. (2018). *Swarm robotics - A formal approach*. Springer. https://doi.org/10.1007/978-3-319-74528-2

Hamilton, W. D. (1963). The evolution of altruistic behavior. *The American Naturalist*, *97*(896), 354–356.

Hamilton, W. D. (1964). The genetical evolution of social behaviour. i & ii. *Journal of theoretical biology*, *7*(1), 17–52.

Hannah Ritchie, M. R., & Rosado, P. (2020). Co2 and greenhouse gas emissions. *Our World in Data*. https://ourworldindata.org/co2-and-other-greenhouse-gas-emissions

Hansell, M. (2007). *Built by animals: the natural history of animal architecture*. OUP Oxford.

Hart, E., Steyven, A., & Paechter, B. (2015). Improving Survivability in Environment-driven Distributed Evolutionary Algorithms through Explicit Relative Fitness and Fitness Proportionate Communication. *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 169–176. https://doi.org/10.1145/2739480.2754688

Harvey, I. (2009). The microbial genetic algorithm. *European Conference on Artificial Life*, 126–133.

Hayat, S., Yanmaz, E., & Muzaffar, R. (2016). Survey on unmanned aerial vehicle networks for civil applications: a communications viewpoint. *IEEE Communications Surveys & Tutorials*, *18*(4), 2624–2661.

Heinerman, J., Drupsteen, D., & Eiben, A. E. (2015). Three-fold Adaptivity in Groups of Robots: The Effect of Social Learning. In S. Silva (Ed.), *Proceedings of the 17th annual conference on genetic and evolutionary computation* (pp. 177–183). ACM.

Heinerman, J., Rango, M., & Eiben, A. E. (2016). Evolution, individual learning, and social learning in a swarm of real robots. *Proceedings - 2015 IEEE Symposium Series on Computational Intelligence, SSCI 2015*, 1055–1062. https://doi.org/10.1109/SSCI.2015.152

Heskes, T., Sijben, E., Bucur, I. G., & Claassen, T. (2020). Causal shapley values: exploiting causal knowledge to explain individual predictions of complex models. *Advances in neural information processing systems*, *33*, 4778–4789.

HolmesParker, C., Agogino, A. K., & Tumer, K. (2013). Exploiting structure and utilizing agent-centric rewards to promote coordination in large multiagent systems. *AAMAS*, 1181–1182.

HolmesParker, C., Agogino, A. K., & Tumer, K. (2016). Combining reward shaping and hierarchies for scaling to large multiagent systems. *The Knowledge Engineering Review*, *31*(1), 3–18.

HolmesParker, C., Taylor, M. E., Agogino, A. K., & Tumer, K. (2014). Clean rewards to improve coordination by removing exploratory action noise. *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, *3*, 127–134.

Horyna, J., Baca, T., & Saska, M. (2021). Autonomous collaborative transport of a beam-type payload by a pair of multi-rotor helicopters. *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, 1139–1147.

Hostallero, D. E., Kim, D., Moon, S., Son, K., Kang, W. J., & Yi, Y. (2020). Inducing cooperation through reward reshaping based on peer evaluations in deep multi-agent reinforcement learning. *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 520–528.

Huber, M. J., & Durfee, E. H. (1995). Deciding when to commit to action during observation-based coordination. *ICMAS*, *95*, 163–170.

Hunt, E. R., & Hauert, S. (2020). A checklist for safe robot swarms. *Nature Machine Intelligence*, *2*(8), 420–422. https://doi.org/10.1038/s42256-020-0213-2

Hunt, S., Meng, Q., Hinde, C., & Huang, T. (2014). A consensus-based grouping algorithm for multi-agent cooperative task allocation with complex requirements. *Cognitive computation*, *6*(3), 338–350.

Iocchi, L., Nardi, D., & Salerno, M. (2000). Reactivity and deliberation: a survey on multi-robot systems. *Workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, 9–32.

Jaques, N., Lazaridou, A., Hughes, E., Gulcehre, C., Ortega, P. A., Strouse, D., Leibo, J. Z., & de Freitas, N. (2018). Social influence as intrinsic motivation for multi-agent deep reinforcement learning.

Juliá, M., Gil, A., & Reinoso, O. (2012). A comparison of path planning strategies for autonomous exploration and mapping of unknown environments. *Autonomous Robots*, *33*(4), 427–444.

Kalai, E., & Lehrer, E. (1993). Rational learning leads to nash equilibrium. *Econometrica: Journal of the Econometric Society*, 1019–1045.

Kandris, D., Nakas, C., Vomvas, D., & Koulouras, G. (2020). Applications of wireless sensor networks: an up-to-date survey. *Applied System Innovation*, *3*(1), 14.

Khamis, A., Hussein, A., & Elmogy, A. (2015). Multi-robot task allocation: a review of the state-of-the-art. *Cooperative robots and sensor networks 2015*, 31–51.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., & Osawa, E. (1997). Robocup: the robot world cup initiative. *Proceedings of the first international conference on Autonomous agents*, 340–347.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., & Matsubara, H. (1997). Robocup: a challenge problem for ai. *AI magazine*, *18*(1), 73–73.

Kitano, H., Tadokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjou, A., & Shimada, S. (1999). Robocup rescue: search and rescue in large-scale disasters as a domain for autonomous agents research. *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 99CH37028)*, *6*, 739–743.

Knudson, M., & Tumer, K. (2010). Coevolution of heterogeneous multi-robot teams. *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, 127–134.

Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: a survey. *The International Journal of Robotics Research*, *32*(11), 1238–1274.

Kok, J. R., Vlassis, N. et al. (2003). Distributed decision making of robotic agents. *Proceedings of the 8th Annual Conference of the Advanced School for Computing and Imaging*, 318–325.

Kolling, A., Sycara, K., Nunnally, S., & Lewis, M. (2013). Human swarm interaction: an experimental study of two types of interaction with foraging swarms. *Journal of Human-Robot Interaction*, *2*(2).

Kraemer, L., & Banerjee, B. (2016). Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, *190*, 82–94.

Krause, A., Singh, A., & Guestrin, C. (2008). Near-optimal sensor placements in gaussian processes: theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, *9*(2).

Kruijff, G.-J. M., Kruijff-Korbayová, I., Keshavdas, S., Larochelle, B., Janíček, M., Colas, F., Liu, M., Pomerleau, F., Siegwart, R., Neerincx, M. A., et al. (2014). Designing, developing, and deploying systems to support human–robot teams in disaster response. *Advanced Robotics*, *28*(23), 1547–1570.

Kube, C., & Bonabeau, E. (2000). Cooperative transport by ants and robots. *Robotics and Autonomous Systems*, *30*(1), 85–101. https://doi.org/https://doi.org/10.1016/S0921-8890(99)00066-4

Kwak, B., Shintake, J., Zhang, L., & Floreano, D. (2022). Towards edible drones for rescue missions: design and flight of nutritional wings. *arXiv preprint arXiv:2211.04149*.

Laurent, G. J., Matignon, L., Fort-Piat, L., et al. (2011). The world of independent learners is not markovian. *International Journal of Knowledge-based and Intelligent Engineering Systems*, *15*(1), 55–64.

Lehman, J., & Stanley, K. O. (2011). Novelty search and the problem with objectives. *Genetic programming theory and practice ix* (pp. 37–56). Springer.

Leibo, J. Z., Hughes, E., Lanctot, M., & Graepel, T. (2019). Autocurricula and the emergence of innovation from social interaction: a manifesto for multi-agent intelligence research. https://doi.org/10.48550/ARXIV.1903.00742

Li, J., Kuang, K., Wang, B., Liu, F., Chen, L., Wu, F., & Xiao, J. (2021). Shapley counterfactual credits for multi-agent reinforcement learning. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery &amp Data Mining*. https://doi.org/10.1145/3447548.3467420

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, 157–163.

Lottes, P., Khanna, R., Pfeifer, J., Siegwart, R., & Stachniss, C. (2017). Uav-based crop and weed classification for smart farming. *2017 IEEE international conference on robotics and automation (ICRA)*, 3024–3031.

Lowe, R., Wu, Y. I., Tamar, A., Harb, J., Pieter Abbeel, O., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, *30*.

Luke, S., Hohn, C., Farris, J., Jackson, G., & Hendler, J. (1997). Co-evolving soccer softbot team coordination with genetic programming. *Robot Soccer World Cup*, 398–411.

Ma, X., Li, X., Zhang, Q., Tang, K., Liang, Z., Xie, W., & Zhu, Z. (2019). A survey on cooperative co-evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, *23*(3), 421–441. https://doi.org/10.1109/TEVC.2018.2868770

Marion, P., Fallon, M., Deits, R., Valenzuela, A., Pérez D'Arpino, C., Izatt, G., Manuelli, L., Antone, M., Dai, H., Koolen, T., et al. (2017). Director: a user interface designed for robot operation with shared autonomy. *Journal of Field Robotics*, *34*(2), 262–280.

Masone, C., Giordano, P. R., Bülthoff, H. H., & Franchi, A. (2014). Semi-autonomous trajectory generation for mobile robots with integral haptic shared control. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 6468–6475.

Matarić, M. J. (1995). Issues and approaches in the design of collective autonomous agents. *Robotics and autonomous systems*, *16*(2-4), 321–331.

Matarić, M. J. (1998). Behavior-based robotics as a tool for synthesis of artificial behavior and analysis of natural behavior. *Trends in cognitive sciences*, *2*(3), 82–86.

Matignon, L., Laurent, G. J., & Le Fort-Piat, N. (2012). Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *The Knowledge Engineering Review*, *27*(1), 1–31.

Matignon, L., Laurent, G. J., Fort-Piat, L., Chapuis, Y.-A., et al. (2010). Designing decentralized controllers for distributed-air-jet mems-based micromanipulators by reinforcement learning. *Journal of intelligent & robotic systems*, *59*(2), 145–166.

Maza, I., Caballero, F., Capitán, J., Martınez-de-Dios, J. R., & Ollero, A. (2011). Experimental results in multi-uav coordination for disaster management and civil security applications. *Journal of intelligent & robotic systems*, *61*(1), 563–585.

McCarragher, B. J. (1993). Robotic assembly and trajectory planning using discrete event modelling. *Proceedings of IEEE 2nd International Workshop on Emerging Technologies and Factory Automation (ETFA'93)*, 188–196.

Meliou, A., Krause, A., Guestrin, C., & Hellerstein, J. M. (2007). Nonmyopic informative path planning in spatio-temporal models. *AAAI*, *10*(4), 16–7.

Mellinger, D., Shomin, M., Michael, N., & Kumar, V. (2013). Cooperative grasping and transport using multiple quadrotors. *Distributed autonomous robotic systems* (pp. 545–558). Springer.

Michael Rubenstein, A. C., & Nagpal, R. (2014). Programmable self-assembly in a thousand-robot swarm. *Science*, *345*(6198), 795–799. https://doi.org/10.1126/science.1254295

Minsky, M. L. (1967). *Computation: finite and infinite machines*. Prentice-Hall, Inc.

Mirhosseini, Y., Zion, M. Y. B., Dauchot, O., & Bredeche, N. (2022). Adaptive phototaxis of a swarm of mobile robots using positive and negative feedback self-alignment. *Proceedings of the Genetic and Evolutionary Computation Conference*, 104–112.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529–533.

Montanier, J.-m., Carrignon, S., & Bredeche, N. (2016). Behavioural Specialization in Embodied Evolutionary Robotics: Why so Difficult ? *Frontiers in Robotics and AI*, 1–17.

Moravčík, M., Schmid, M., Burch, N., Lisý, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., & Bowling, M. (2017). Deepstack: expert-level artificial intelligence in heads-up no-limit poker. *Science*, *356*(6337), 508–513.

Mountz, M. C., D'andrea, R., Laplante, J. A., Lyons, I. D. P., Mansfield, P. K., & Amsbury, B. W. (2008). Inventory system with mobile drive unit and inventory holder [US Patent 7,402,018].

Moussaïd, M., Helbing, D., & Theraulaz, G. (2011). How simple rules determine pedestrian behavior and crowd disasters. *Proceedings of the National Academy of Sciences*, *108*(17), 6884–6888. https://doi.org/10.1073/pnas.1016507108

Murray, R. M. (2007). Recent research in cooperative control of multivehicle systems. *Journal of Dynamic Systems, Measurement, and Control*, *129*(5), 571–583.

Naghsh, A. M., Gancet, J., Tanoto, A., & Roast, C. (2008). Analysis and design of human-robot swarm interaction in firefighting. *RO-MAN 2008-The 17th IEEE Inter-*

national Symposium on Robot and Human Interactive Communication, 255–260.

Neumann, P. P., Asadi, S., Lilienthal, A. J., Bartholmai, M., & Schiller, J. H. (2012). Autonomous gas-sensitive microdrone: wind vector estimation and gas distribution mapping. *IEEE robotics & automation magazine*, *19*(1), 50–61.

Ng, A. Y., Harada, D., & Russell, S. J. (1999). Policy invariance under reward transformations: theory and application to reward shaping. *Proceedings of the Sixteenth International Conference on Machine Learning*, 278–287.

Nilsson, N. J. et al. (1984). Shakey the robot.

Nitschke, G. (2012). Behavioral heterogeneity, cooperation, and collective construction. *2012 IEEE Congress on Evolutionary Computation*, 1–8.

Noë, R., & Hammerstein, P. (1994). Biological markets: supply and demand determine the effect of partner choice in cooperation, mutualism and mating. *Behavioral ecology and sociobiology*, *35*(1), 1–11.

Nolfi, S., & Floreano, D. (2000). *Evolutionary robotics: the biology, intelligence, and technology of self-organizing machines*. MIT press.

Nouyan, S., Campo, A., & Dorigo, M. (2008). Path formation in a robot swarm. *Swarm Intelligence*, *2*(1), 1–23.

Olfati-Saber, R. (2006). Flocking for multi-agent dynamic systems: algorithms and theory. *IEEE Transactions on automatic control*, *51*(3), 401–420.

Olfati-Saber, R., Fax, J. A., & Murray, R. M. (2007). Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, *95*(1), 215–233.

Oliehoek, F. A., Spaan, M. T., & Vlassis, N. (2008). Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research*, *32*, 289–353.

Ollero, A., Lacroix, S., Merino, L., Gancet, J., Wiklund, J., Remuß, V., Perez, I. V., Gutiérrez, L. G., Viegas, D. X., Benitez, M. A. G., et al. (2005). Multiple eyes in the skies: architecture and perception issues in the comets unmanned air vehicles project. *IEEE robotics & automation magazine*, *12*(2), 46–57.

Panait, L. (2010). Theoretical convergence guarantees for cooperative coevolutionary algorithms. *Evol. Comput.*, *18*(4), 581–615. https://doi.org/10.1162/EVCO_a_00004

Panait, L., & Tuyls, K. (2007). Theoretical advantages of lenient q-learners: an evolutionary game theoretic perspective. *Proceedings of the 6th international joint conference on autonomous agents and multiagent systems*, 1–8.

Parker, L. E. (1998). Alliance: an architecture for fault tolerant multirobot cooperation. *IEEE transactions on robotics and automation, 14*(2), 220–240.

Parker, L. E. (2007). Distributed intelligence: overview of the field and its application in multi-robot systems. *AAAI fall symposium: regarding the intelligence in distributed intelligent systems*, 1–6.

Parker, L. E. (2008). Multiple Mobile Robot Systems. In S. B. Heidelberg (Ed.), *Handbook of robotics* (pp. 921–941). Springer.

Parker, L. E., Rus, D., & Sukhatme, G. S. (2016). Multiple mobile robot systems. In B. Siciliano & O. Khatib (Eds.), *Springer handbook of robotics* (pp. 1335–1384). Springer International Publishing. https://doi.org/10.1007/978-3-319-32552-1_53

Parkes, D. C. (2004). On learnable mechanism design. *Collectives and the design of complex systems* (pp. 107–131). Springer.

Pennisi, E. (2005). How did cooperative behavior evolve? *Science, 309*(5731), 93–93. https://doi.org/10.1126/science.309.5731.93

Perolat, J., Piot, B., & Pietquin, O. (2018). Actor-critic fictitious play in simultaneous move multistage games. *International Conference on Artificial Intelligence and Statistics*, 919–928.

Perolat, J., Vylder, B. D., Hennes, D., Tarassov, E., Strub, F., de Boer, V., Muller, P., Connor, J. T., Burch, N., Anthony, T., McAleer, S., Elie, R., Cen, S. H., Wang, Z., Gruslys, A., Malysheva, A., Khan, M., Ozair, S., Timbers, F., … Tuyls, K. (2022). Mastering the game of stratego with model-free multiagent reinforcement learning. *Science, 378*(6623), 990–996. https://doi.org/10.1126/science.add4679

Petersen, K. H., Napp, N., Stuart-Smith, R., Rus, D., & Kovac, M. (2019). A review of collective robotic construction. *Science Robotics, 4*(28), eaau8479.

Peterson, J. L. (1977). Petri nets. *ACM Computing Surveys (CSUR), 9*(3), 223–252.

Potter, M. A., & De Jong, K. A. (1994a). A cooperative coevolutionary approach to function optimization. In Y. Davidor, H.-P. Schwefel, & R. Männer (Eds.), *Parallel problem solving from nature — ppsn iii* (pp. 249–257). Springer Berlin Heidelberg.

Potter, M. A., & De Jong, K. A. (1994b). A cooperative coevolutionary approach to function optimization. *International Conference on Parallel Problem Solving from Nature*, 249–257.

Potter, M. A., & De Jong, K. A. (2000). Cooperative coevolution: an architecture for evolving coadapted subcomponents. *Evolutionary Computation, 8*, 1–29.

Potter, M. A., Meeden, L. A., Schultz, A. C., et al. (2001). Heterogeneity in the coevolved behaviors of mobile robots: the emergence of specialists. *International joint conference on artificial intelligence*, *17*(1), 1337–1343.

Prieto, A., Bellas, F., Faina, A., & Duro, R. (2009). Asynchronous Situated Coevolution and Embryonic Reproduction as a Means to Autonomously Coordinate Robot Teams. *Knowledge-based and intelligent information and engineering systems se - 43* (pp. 351–359). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-04595-0{_}43

Queralta, J. P., Taipalmaa, J., Pullinen, B. C., Sarker, V. K., Gia, T. N., Tenhunen, H., Gabbouj, M., Raitoharju, J., & Westerlund, T. (2020). Collaborative multi-robot search and rescue: planning, coordination, perception, and active vision. *Ieee Access*, *8*, 191617–191643.

Quinn, M., Smith, L., Mayley, G., & Husbands, P. (2002). Evolving formation movement for a homogeneous multi-robot system: teamwork and role-allocation with real robots. *COGNITIVE SCIENCE RESEARCH PAPER-UNIVERSITY OF SUSSEX CSRP*.

Rahmattalabi, A., Chung, J. J., Colby, M., & Tumer, K. (2016). D++: structural credit assignment in tightly coupled multiagent domains. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4424–4429.

Rashid, T., Farquhar, G., Peng, B., & Whiteson, S. (2020). Weighted qmix: expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *Advances in neural information processing systems*, *33*, 10199–10210.

Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., & Whiteson, S. (2018). Qmix: monotonic value function factorisation for deep multi-agent reinforcement learning. *International Conference on Machine Learning*, 4295–4304.

Reif, J. H., & Wang, H. (1999). Social potential fields: a distributed behavioral control for autonomous robots. *Robotics and Autonomous Systems*, *27*(3), 171–194. https://doi.org/https://doi.org/10.1016/S0921-8890(99)00004-4

Reynaud, L., & Guérin-Lassous, I. (2016). Physics-based swarm intelligence for disaster relief communications. *International Conference on Ad-Hoc Networks and Wireless*, 93–107.

Rizk, Y., Awad, M., & Tunstel, E. W. (2019). Cooperative heterogeneous multi-robot systems: a survey. *ACM Comput. Surv.*, *52*(2). https://doi.org/10.1145/3303848

Robinson, D., & Goforth, D. (2005). *The topology of the 2x2 games: a new periodic table* (Vol. 3). Psychology Press.

Rockefeller, G., Khadka, S., & Tumer, K. (2020). Multi-level fitness critics for cooperative coevolution. *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020)*, 1143–1151. https://par.nsf.gov/biblio/10197808

Roldán-Gómez, J. J., González-Gironda, E., & Barrientos, A. (2021). A survey on robotic technologies for forest firefighting: applying drone swarms to improve firefighters' efficiency and safety. *Applied Sciences*, *11*(1), 363.

Rosser Jr, J. C., Vignesh, V., Terwilliger, B. A., & Parker, B. C. (2018). Surgical and medical applications of drones: a comprehensive review. *JSLS: Journal of the Society of Laparoendoscopic Surgeons*, *22*(3).

Roughgarden, T. (2010). Algorithmic game theory. *Communications of the ACM*, *53*(7), 78–86.

Rubenstein, M., Ahler, C., & Nagpal, R. (2012). Kilobot: a low cost scalable robot system for collective behaviors. *Proceedings - IEEE International Conference on Robotics and Automation*, 3293–3298. https://doi.org/10.1109/ICRA.2012.6224638

Rubenstein, M., Cornejo, A., & Nagpal, R. (2014). Programmable self-assembly in a thousand-robot swarm. *Science*, *345*(6198), 795–799.

Russell, S. J. (2010). *Artificial intelligence a modern approach*. Pearson Education, Inc.

Santos, M., & Egerstedt, M. (2021). From motions to emotions: can the fundamental emotions be expressed in a robot swarm? *International Journal of Social Robotics*, *13*(4), 751–764.

Santos, M., Notomista, G., Mayya, S., & Egerstedt, M. (2020). Interactive multi-robot painting through colored motion trails. *Frontiers in Robotics and AI*, *7*, 580415.

Schmickl, T., & Hamann, H. (2011). Beeclust: a swarm algorithm derived from honeybees. *Bio-inspired computing and communication networks*, 95–137.

Schulz, A. S., & Stier Moses, N. E. (2002). On the performance of user equilibrium in traffic networks.

Schwab, K. (2017). *Fourth industrial revolution*.

Shahrokhi, S., & Becker, A. T. (2016). Object manipulation and position control using a swarm with global inputs. *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, 561–566.

Shakhatreh, H., Sawalmeh, A. H., Al-Fuqaha, A., Dou, Z., Almaita, E., Khalil, I., Othman, N. S., Khreishah, A., & Guizani, M. (2019). Unmanned aerial vehicles (uavs): a survey on civil applications and key research challenges. *Ieee Access*, *7*, 48572–48634.

Shapley, L. S. (1953a). Stochastic games*. *Proceedings of the National Academy of Sciences*, *39*(10), 1095–1100. https://doi.org/10.1073/pnas.39.10.1095

Shapley, L. S. (1953b). A value for n-person games (H. W. Kuhn & A. W. Tucker, Eds.). *Contributions to the Theory of Games II (Annals of Mathematics Studies 28)*, 307–317.

Shoham, Y., & Leyton-Brown, K. (2008a). *Multiagent systems: algorithmic, game-theoretic, and logical foundations*. Cambridge University Press.

Shoham, Y., & Leyton-Brown, K. (2008b). *Multiagent systems: algorithmic, game-theoretic, and logical foundations*. Cambridge University Press.

Siciliano, B., Khatib, O., & Kröger, T. (2008). *Springer handbook of robotics* (Vol. 200). Springer.

Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). *Introduction to autonomous mobile robots*. MIT press.

Silva, F., Urbano, P., Oliveira, S., & Christensen, A. L. (2012). odNEAT: An Algorithm for Distributed Online, Onboard Evolution of Robot Behaviours. *Artificial Life 13*, 251–258. https://doi.org/10.7551/978-0-262-31050-5-ch034

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

Singh, A., Krause, A., Guestrin, C., & Kaiser, W. J. (2009). Efficient informative sensing using multiple robots. *Journal of Artificial Intelligence Research*, *34*, 707–755.

Slavkov, I., Carrillo-Zapata, D., Carranza, N., Diego, X., Jansson, F., Kaandorp, J., Hauert, S., & Sharpe, J. (2018). Morphogenesis in robot swarms. *Science Robotics*, *3*(25), eaau9178. https://doi.org/10.1126/scirobotics.aau9178

Son, K., Kim, D., Kang, W. J., Hostallero, D. E., & Yi, Y. (2019). Qtran: learning to factorize with transformation for cooperative multi-agent reinforcement learning. *International conference on machine learning*, 5887–5896.

Soria, E., Schiano, F., & Floreano, D. (2021). Predictive control of aerial swarms in cluttered environments. *Nature Machine Intelligence*, *3*(6), 545–554.

Soysal, O., & Şahin, E. (2006). A macroscopic model for self-organized aggregation in swarm robotic systems. *International Workshop on Swarm Robotics*, 27–42.

Spears, W. M., Spears, D. F., Hamann, J. C., & Heil, R. (2004). Distributed, physics-based control of swarms of vehicles. *Autonomous robots*, *17*(2), 137–162.

Stone, P., Kaminka, G. A., Kraus, S., & Rosenschein, J. S. (2010). Ad hoc autonomous agent teams: collaboration without pre-coordination. *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 1504–1509.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., & Graepel, T. (2018). Value-decomposition networks for cooperative multi-agent learning based on team reward. *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 2085–2087.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning, second edition: an introduction*. MIT Press.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems, 12*.

Tagliabue, A., Kamel, M., Siegwart, R., & Nieto, J. (2019). Robust collaborative object transportation using multiple mavs. *The International Journal of Robotics Research*, *38*(9), 1020–1044.

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., & Vicente, R. (2017). Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, *12*(4), e0172395.

Tan, M. (1993). Multi-agent reinforcement learning: independent vs. cooperative agents. *In Proceedings of the Tenth International Conference on Machine Learning*, 330–337.

Tarapore, D., Floreano, D., & Keller, L. (2006). Influence of the level of polyandry and genetic architecture on division of labour. *Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems*, (CONF), 358–364.

Teh, X. X., Aijaz, A., Portelli, A., & Jones, S. (2020). Communications-based formation control of mobile robots: modeling, analysis and performance evaluation. *Proceedings of the 23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 149–153.

Tesauro, G. (1994). Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, *6*(2), 215–219.

Tesauro, G. (2003). Extending q-learning to general adaptive multi-agent systems. *Advances in neural information processing systems*, *16*.

Tokekar, P., Branson, E., Vander Hook, J., & Isler, V. (2013). Tracking aquatic invaders: autonomous robots for monitoring invasive fish. *IEEE Robotics & Automation Magazine*, *20*(3), 33–41.

Tokekar, P., Vander Hook, J., Mulla, D., & Isler, V. (2016). Sensor planning for a symbiotic uav and ugv system for precision agriculture. *IEEE Transactions on Robotics*, *32*(6), 1498–1511.

Trianni, V. (2008). *Evolutionary swarm robotics: evolving self-organising behaviours in groups of autonomous robots* (Vol. 108). Springer.

Trianni, V., & Dorigo, M. (2006). Self-organisation and communication in groups of simulated and physical robots. *Biological cybernetics*, *95*(3), 213–231.

Trianni, V., Groß, R., Labella, T. H., Şahin, E., & Dorigo, M. (2003). Evolving aggregation behaviors in a swarm of robots. *European Conference on Artificial Life*, 865–874.

Trianni, V., Nolfi, S., & Dorigo, M. (2008). Evolution, Self-organization and Swarm Robotics. *Swarm intelligence* (pp. 163–191). https://doi.org/10.1007/978-3-540-74089-6{_}5

Tumer, K., Agogino, A. K., & Wolpert, D. H. (2002). Learning sequences of actions in collectives of autonomous agents. *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, 378–385.

Turner, K. (2006). Designing agent utilities for coordinated, scalable and robust multi-agent systems. *Coordination of large-scale multiagent systems* (pp. 173–188). Springer.

Tuyls, K., & Stone, P. (2017). Multiagent learning paradigms. *EUMAS/AT*.

Uzakov, T., Nascimento, T. P., & Saska, M. (2020). Uav vision-based nonlinear formation control applied to inspection of electrical power lines. *2020 international conference on unmanned aircraft systems (ICUAS)*, 1301–1308.

Valentini, G., Brambilla, D., Hamann, H., & Dorigo, M. (2016). Collective perception of environmental features in a robot swarm. *International Conference on Swarm Intelligence*, 65–76.

Valentini, G., Ferrante, E., & Dorigo, M. (2017). The best-of-n problem in robot swarms: formalization, state of the art, and novel perspectives. *Frontiers in Robotics and AI*, *4*, 9.

Valentini, G., Ferrante, E., Hamann, H., & Dorigo, M. (2016a). Collective decision with 100 kilobots: speed versus accuracy in binary discrimination problems. *Autonomous Agents and Multi-Agent Systems*, *30*(3), 553–580. https://doi.org/10.1007/s10458-015-9323-3

Valentini, G., Ferrante, E., Hamann, H., & Dorigo, M. (2016b). Collective decision with 100 kilobots: speed versus accuracy in binary discrimination problems. *Autonomous agents and multi-agent systems*, *30*(3), 553–580.

Veruggio, G., Operto, F., & Bekey, G. (2016). Roboethics: social and ethical implications. *Springer handbook of robotics* (pp. 2135–2160). Springer.

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

Waibel, M., Keays, B., & Augugliaro, F. (2017). *Drone shows: creative potential and best practices* (tech. rep.). ETH Zurich.

Waibel, M., Keller, L., & Floreano, D. (2009). Genetic team composition and level of selection in the evolution of cooperation. *IEEE transactions on Evolutionary Computation*, *13*(3), 648–660.

Wang, Z., & Schwager, M. (2016a). Force-amplifying n-robot transport system (force-ants) for cooperative planar manipulation without communication. *The International Journal of Robotics Research*, *35*(13), 1564–1586.

Wang, Z., & Schwager, M. (2016b). Kinematic multi-robot manipulation with no communication using force feedback. *2016 IEEE international conference on robotics and automation (ICRA)*, 427–432.

Watkins, C. J. C. H. (1989). Learning from delayed rewards.

Watson, R. A., Ficici, S. G., & Pollack, J. B. (2002). Embodied Evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, *39*(1), 1–18. https://doi.org/10.1016/S0921-8890(02)00170-7

Wawerla, J., Sukhatme, G. S., & Mataric, M. J. (2002). Collective construction with multiple robots. *IEEE/RSJ international conference on intelligent robots and systems*, *3*, 2696–2701.

Werfel, J., & Nagpal, R. (2006). Extended stigmergy in collective construction. *IEEE Intelligent Systems*, *21*(2), 20–28.

Werger, B. B., & Matarić, M. J. (2000). Broadcast of local eligibility for multi-target observation. *Distributed autonomous robotic systems 4* (pp. 347–356). Springer.

West, S. A., Griffin, A. S., & Gardner, A. (2007). Social semantics: altruism, cooperation, mutualism, strong reciprocity and group selection. *Journal of evolutionary biology*, *20*(2), 415–32. https://doi.org/10.1111/j.1420-9101.2006.01258.x

Wolpert, D., Tumer, K., & Swanson, K. (2000). Optimal wonderful life utility functions in multi-agent systems.

Wolpert, D. H., & Tumer, K. (2001). Optimal payoff functions for members of collectives. *Advances in Complex Systems*, *4*(2/3), 265–279.

Wolpert, D. H., & Tumer, K. (2008). *An introduction to collective intelligence* (tech. rep.). NASA.

Wurman, P. R., D'Andrea, R., & Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, *29*(1), 9–9.

Yamauchi, B. (1997). A frontier-based approach for autonomous exploration. *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*, 146–151.

Yamauchi, B. (1998). Frontier-based exploration using multiple robots. *Proceedings of the second international conference on Autonomous agents*, 47–53.

Yun, S.-k., & Rus, D. (2007). Optimal distributed planning of multi-robot placement on a 3d truss. *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1365–1370.

Yun, S.-k., Schwager, M., & Rus, D. (2011). Coordinating construction of truss structures using distributed equal-mass partitioning. *Robotics research* (pp. 607–623). Springer.

Zerbel, N., & Tumer, K. (2020). The power of suggestion. *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020)*, 1602–1610.

Zhang, C., Noguchi, N., & Yang, L. (2016). Leader–follower system using two robot tractors to improve work efficiency. *Computers and Electronics in Agriculture*, *121*, 269–281.

Zhang, K., Yang, Z., & Başar, T. (2019). Multi-agent reinforcement learning: a selective overview of theories and algorithms.

Zhang, Q., Lu, C., Garg, A., & Foerster, J. (2021). Centralized model and exploration policy for multi-agent rl. *arXiv preprint arXiv:2107.06434*.

Zion, M. Y. B., Fersula, J., Bredeche, N., & Dauchot, O. (2021). Morphological computation and decentralized learning in a swarm of sterically interacting robots. https://doi.org/10.48550/ARXIV.2111.06953

Zlot, R., & Stentz, A. (2006). Market-based multirobot coordination for complex tasks. *The International Journal of Robotics Research, 25*(1), 73–101.