Distributed Embodied Evolutionary Adaptation of Behaviors in Swarms of Robotic Agents

Iñaki Fernández Pérez

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Distributed Embodied Evolutionary Adaptation of Behaviors in Swarms of Robotic Agents

THÈSE

présentée et soutenue publiquement le 19 décembre 2017

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Iñaki FERNÁNDEZ PÉREZ

Composition du jury

Rapporteurs :	Nicolas BREDÈCHE	Professeur UPMC et ISIR, Paris, France
	Evert HAASDIJK	Assistant professor Vrije Universiteit Amsterdam, Pays-Bas
Examinateurs :	Jean-Louis DENEUBOURG	Chercheur F.R.S. Université Libre de Bruxelles, Belgique
	Irina ILLINA	Maître de conférences Université de Lorraine, Nancy, France
Directeurs :	François CHARPILLET	Directeur de recherche Inria Nancy, France
	Amine BOUMAZA	Maître de conférences Université de Lorraine, Nancy, France

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Loria
Laboratoire lorrain de recherche en informatique et ses applications

Mis en page avec la classe thesul.

Acknowledgments

Abro hilo. I do not want to fall into stereotypes, but it has been a long road until I arrived here. Of course, there is still much to come, but I feel that it is a good moment to turn back a little bit, and remember all the people that had an impact in my life, without whom I would not be here now. There is not any particular order, and without a doubt, I will forget many. Worry not: they are not forgotten in my heart. Here I go.

I want to thank my supervisors. Amine, you are probably the person that has taught me the most in terms of science, and from you I got the itch for research. François, our interactions got progressively more into depth, you provided a splendid environment for me to work, and I will always admire your ever-present ability to take a step back and ask "yeah, but, what does this mean?". I also want to thank the members of my thesis committee. I thank my reviewers, Nicolas and Evert, not only for accepting to review this thesis, but also for their remarks and suggestions on my work, which significantly helped to improve it, and for being source of inspiration during our interactions over the years. I thank my examiners, Jean-Louis Deneubourg and Irina Illina, for accepting to participate in this committee.

The people in both MaIA and LARSEN teams, the people with whom I shared office, my other colleagues in the lab, you all were a great support, as well as source of inspiration and help when I needed it. Thank you Nassim, Mihai, Quan, Benjamin, Jilles, Amandine, Mauricio, Son, Vincent, Matthieu, Arsène, Jano, Nicolas, Mélanie, Thomas, Vassilis, Adrian, Yassine, Adrien, Abdallah, Kazu, Roberto, Costas, Jonathan, Oriane, Dorian, Francis, Serena, JB, Karim, Alain, Olivier, Valerio, Jamie, Adam, Pauline, Maxime, Federico, Sebastián, Francesco, Théo, Alexandre, Thomas, Amédeo, Emmanuel, Yannick, Adrien, Nicolas, Émilie, and other people I probably forgot. I want to give a warm thank you to the assistants that managed to do an amazing work with all the paperwork we all hate: thank you Véronique, Laurence, Blandine, Françoise, Virginie, Sylvie. Another special thank you goes to my colleagues when teaching, both at IUT Charlemagne, and at UFR Math-Info: thank you Gérôme, Yolande, Slim, Samuel, Vincent, Nadia, Amine, Geoffray, Alexandre, Romain, Armelle, Gilles, Maxime, Manuel, Olivier, Antoine, Christine, Amaury, Laurent.

A very special thank you goes to you, the one that was always there. Without you, I would never have become myself, and I would never have lasted so long: to the MUSIC. You kept me sane, gave me the greatest smiles and the best satisfactions ever. You still do. And with you, there is all the people with whom I shared such moments: Bo & The Gas, with Víctor, Chechu, Pablo, Raúl, Elena, Luis, then Die Barrikade, with Colom, Merino, Piti, and the army. Then, in Nancy, Martín, Dani, Víctor (ch), Mauricio, Franchement Fantastico (sorry guys, I cannot list you all! We were legion!), the post-FF with Bibi and the rest, BIG JAZ, with Benoît, Adrien, Jirka, Yann, Baldwin et Sylvain, then Léa, yeah, ever-diva Léa, through BJN (also with Jérém and then Jean), but especially with Hat Stufff (you both, Léa and Adrian), great times, great music, great concerts, great compositions. Also, I want to thank the ones that I found during my last year, however short it was: Ken and Diego (insane structures: next time, just throw a dice), but also Kamel, nice discovery with the *chaâbi* music (الموسيقى الشعبية). To this list, I want to add my first teachers, Fernando, Sergio, Ángel, and the people from the *Banda Municipal* from La Cistérniga. On another topic, this thesis would not have been the same without the energy I got from several sources: thank you Murat (*La maison*), Adil (*La rose*), Isabelle, Caroline, Tarek (*La cantine*), Deniz, Murat, Margaux (*Le 23*), Omar, Jamal, and the others (*Resto*), Saïdi, Salah (*Méditerranéen*).

I want to thank the ones that were there during my "inception" back in Valladolid, old friends,

classmates and other people. Eli, Chino, Mery, Nacho, Mao, H, Fonsi, Eros, Rumba, Cris, Mario, Arota, David, Pitu, Pimiento, and so many more people, I had great times. Mario, Rubén, Vero, Diego, Chuchi, Sanjo, Adri, Pedro, and other people from *Delicias*, I owe you great deal. My classmates at Valladolid University: Garrosa, Travi, Sastre, Sandín, Luis, and all the rest, we learned many things together. I also want to thank my other friends and journey fellows since I arrived at Nancy, almost eight years ago. There were the people during my Erasmus: Bea, Cris, Gaby, Marce, Jo, Chito, Marta, Dom, Anya, Lian, great times guys! Also, the post-Erasmus people: Lautaro, Thomas, Hernán, Vero and the rest, Ioanna, Xavier, Nico, Thao, Chedy, and others. All the pages in this manuscript would not be enough to say what I owe to Laura, with whom I shared so many moments: the good, the bad, and the ugly. Thanks *bicho*!. I then started my M.Sc., and so many other nice guys came along, with whom I learned a lot, both about CS and life: Pierre, Jean, Élian, Thomas, Alexandre, Othman, Pierre-Jean, Kristal, Felipe, Romain, and so many others. I also want to thank my teachers and professors along the years, who knew how to share their knowledge and show it as what it is: the most invaluable richness in the world. I thus thank Tomás, Manolo, Isabel, Isidro, Pilar, Nuria, Félix, Carlos, Benjamín, Javier, José Manuel, César, Arancha, Bernard, Horatiu, Didier, Dominique, Dmitry, Odile, Marie-Odile, Laurent, Adrien, Vincent (Chevrier), Vincent (Thomas), Nazim, Sylvain, and many more throughout the years.

Finally, I thank my family, especially my parents, Luisito and Maite, for providing with splendid means for my education, being patient when they should, and not so much when they had to. The main reason I thank them for, is for fostering and nurturing my curiosity and my quest for knowledge. Looking backwards, it is undoubtedly the one thing that shaped the most my scientific journey and my current life. Thanks also to Geni, Carlitos, Paloma, Manolo, Alicia, Begoña, Toni, Nano, Iván, Sandra, Azahara, Marisa, Ángela (sorry, Mari Ángeles), María, Maruja, Antonio, Luis, Chelo, Tinín. I want to thank all the cats I ever had, for the feeling of peace and the company they always provided: Tapón, Coco, Florita, and especially Rojillo, that has been with me during this Ph.D., waking me up in the middle of the night, just for the fun of doing it.

I am going to leave it here, because otherwise I would never stop. Here goes to such an amazing bunch of people:

Gracias Totales

Sine ore loquens, dominatum in animum exercet.
Ancient latin proverb

To everyone I have ever met.

À tous ceux que j'ai rencontré.

A todos con los que me he cruzado.

Contents

Part II Distributed Adaptation of Swarm Robot Behavior

Finale

Context and Overview

1

Introduction

Contents

This dissertation describes our research on distributed artificial evolution to adapt behaviors in swarms of robotic agents. In Nature, we find collective systems that display complex behaviors by following simple rules. Flocks of birds fly in formation without a leader by locally interacting with each other, and fish schools are able to efficiently coordinate to avoid predators without resorting to any complex mechanism. Additionally, natural systems adapt to their environment by progressively modifying their behavior with experience to improve their performance. For example, evolution provides a means for biological organisms to adapt over generations to improve reproductive success. Collective systems in Nature may need to be capable of such adaptation, to better fit unknown and possibly changing environments. Our work falls at the crossroads of Complex Systems, which investigates the behaviors of collective systems, and Machine Learning, which investigates artificial systems that improve their performance with experience.

The field of Complex Systems investigates large groups of simple entities that interact to give rise to emerging global phenomena. More specifically, Swarm Intelligence is a subfield of Complex Systems that takes inspiration from collective systems in biology, such as behavioral models of social insects, to solve given problems. In our work, we consider large groups of robotic agents that interact in their environment to solve given tasks. This is known as Swarm Robotics, which belongs to Swarm Intelligence. In our case, each entity in the swarm is a robotic agent that perceives its local environment using sensors, and acts upon the environment using motors. The design of behaviors in swarms of robots is a challenging problem, to which few methodologies exist. This challenge arises from the fact that the problem of finding local rules to give rise to global solutions for a given problem is difficult[1].

In our work, we use Machine Learning to automatically adapt the behaviors in swarms of robots. Machine Learning approaches aim at improving the performance of a system using feedback from

[1] Conversely, the inverse problem of observing the global result of given local rules is much more straightforward.

data or experience. Such approaches are usually divided in different families, regarding the type of feedback used for systems to learn. This may include labeled or unlabeled data examples, or reward signals for an agent interacting with its environment. In our work, we particularly focus on adapting agents, which feature a specific type of learning. Such agents exploit feedback from experience as it becomes available, *i.e.* online, which allows for a gradual modification to better fit unknown and possibly changing conditions (*adaptation*). Specifically, we use Artificial Evolution approaches, which take inspiration from evolutionary theory to gradually optimize solutions for a given search problem, in order to adapt behaviors in swarms of robotic agents.

In this thesis, we investigate algorithms that belong to the distributed Embodied Evolutionary Robotics paradigm. In these approaches, robots in a swarm gradually adapt their behaviors using Evolutionary Algorithms. These algorithms are run in parallel by each robot (onboard) while operating for the actual task (online). Further, robots communicate with each other when meeting to share information about their adaptation; as such, adaptation is distributed over the swarm, emerging from local interactions between different learning robots. This could be seen as a type of social learning for robots in the swarm. In these approaches, adaptation is done during actual execution of the behaviors, it is asynchronous between different robots in the swarm, and it is distributed, thus not requiring a central entity that orchestrates evolution. These swarms of interacting robots constitute complex systems that adapt over time to unknown and possibly changing environments and problems. In our work, we aim at better understanding and improving behavior adaptation in such swarms of robotic agents that use distributed evolutionary approaches.

1.1 Context

Collective Adaptive Systems (CASs) refers to systems that consist of large numbers of entities that "*provide more functionality when they are coupled*" (collective) [Kernbach *et al.*, 2011], and that "*change their control rules with experience*" (adaptive) [Anderson *et al.*, 2013], *e.g.* in pursuit of an individual or collective goal. Regarding systems made of a large number of entities, Swarm Intelligence (SI) [Bonabeau *et al.*, 1999] is the research field that aims at designing decentralized and self-organized populations of simple agents. Research in SI mainly takes inspiration from biological swarms, like social (and eusocial) insect colonies (ants, bees, ...), fish schools or bird flocks. In these natural swarms, often some global complex behavior emerges from the interactions between the members of the swarm, while each individual applies simple rules and may not be aware of such global patterns [Deneubourg and Goss, 1989]. For instance, some species of ants are able to forage food by collectively depositing pheromones in the path toward the food source. This, in turn, creates a pheromone gradient that allows other ants in the colony to navigate to that food source. Each individual ant follows rather simple local rules (locally following pheromone gradients and depositing pheromones), but the interactions between ants and with the environment make it possible for the ant colony to locate and forage food collectively.

The application of SI approaches to multirobot systems is known as Swarm Robotics (SR) [Şahin and Spears, 2004]. In SR systems, the focus is generally set on the collective behavior emerging from the interactions between individual robots, rather than aiming at achieving complex behaviors at the individual level. Robotic swarms are thus multirobot teams composed of large numbers of simple robots that operate in a decentralized and self-organized manner.

Engineering of swarm robot control systems is a difficult task for several reasons. Individual robots are complex, have many degrees of freedom, their components (such as sensors and motors) are noisy, and thus there is always uncertainty or inaccuracy in the measurements and movements. Additionally, robot swarms increase the complexity of the system, due to interactions among

robots and decentralized control, making engineering of control in such systems an even harder task. As such, automatic approaches to the synthesis of robot, multirobot, and swarm robot control programs are of utmost interest. Most of the existing work on this topic falls into the general field of Machine Learning (ML) [Mitchell, 1997], and deals with building robot control systems from data examples or experience.

When robotic swarms are deployed in unknown and possibly changing environments, the design of their behavior becomes an even harder problem. In such environments, robot autonomy is not only advisable but crucial. These environments make it unfeasible to foresee every possible situation faced by the robots, and, usually, human guidance is either extremely costly, delayed, or simply not possible. Given these limitations regarding human supervision, and since fewer *a priori* assumptions can be made with respect to the environment in which robots will operate, designing autonomous robot behavior poses additional challenges. In this case, robots could be programmed to robustly cope with a wide range of possible situations, by being either sufficiently general or by including a large repertoire of different task-specific control systems. However, this could be extremely impractical or even impossible, particularly since the number of different situations a robot can experience is potentially huge, and not all of them can be predicted. Additionally, the robot's environment can dynamically and unexpectedly change during its operation, which further complexifies the design of robotic systems.

In our work, we take an agent-based perspective to design behaviors for a swarm of robots. An agent can be defined as an entity that is situated in a possibly changing environment, and interacts with it using sensing and acting capabilities. The physical structure, along with sensing and acting capabilities of an agent, are known as the agent's morphology. Given a certain morphology, such robotic agents can already physically perform a wide variety of tasks, provided that they have the right control or decision-making capabilities. In our work, we consider robots as physically grounded (or embodied) agents that interact with the real world by acquiring local information through their sensors and acting on the environment using their motors. The agent paradigm allows to reason about autonomous robot control at a higher level of abstraction, in which some features about physical robot systems are sidestepped, and delegated to other modules. This allows for designers to focus on the logic of the robot to solve the task, instead of focusing on the implementation details, such as sensor and motor calibration.

In this thesis, we investigate adaptive mechanisms for robot control in swarms of autonomous robotic agents, which are large sets of simple robots that need to adapt to an unknown environment when learning to solve given tasks. Typically, control in robot swarms is decentralized, where each robot is in charge of executing its actions to fulfill a given global goal. The cost of these simple robots is steadily decreasing, while their reliability and sophistication is increasing over time; this makes robot swarms a compelling choice in robotics. However, designing control for such collective systems is challenging, and the existing theoretical frameworks are limited.

Specifically, we are concerned with the adaptation of swarm robot control to address tasks in given environments. Such mechanisms for adaptation through experience fall in the field of Machine Learning, which is discussed in Chapter 2. Robot morphology design is out of the scope of this work, and, in the remainder of this thesis, fixed-morphology homogeneous robot swarms are assumed. Concretely, we conducted a set of studies using distributed Evolutionary Algorithms (EAs), run onboard by each robot during operation (*i.e.* online), to automatically synthesize and adapt neurocontrollers for swarms of robotic agents, to solve given tasks. These techniques are known as distributed Embodied Evolution (EE) algorithms [Watson *et al.*, 2002, Eiben *et al.*, 2010a], and they exploit the intrinsic parallelism of robots in a swarm to improve learning. Indeed, in these approaches, robots locally exchange information, *e.g.* good quality controllers, to help each other learn. In this sense, global learning emerges from both individual learning for each robot

and local exchanges of the results of learning among nearby robots, which can be considered as a "bottom-up" approach to collective learning. This would be opposed to a "top-down" approach to collective learning, *e.g.* one in which a local process of learning is derived from global specifications.

1.2 Autonomous Robots, Agents, and Swarms

Autonomy in robotics refers to the extent to which a robot is able to work for long periods without human intervention [From *et al.*, 2016]. This property of a robotic system does not draw a clear line between what an autonomous robot is and what it is not. This should be seen more as a continuum of autonomy: from very dependent robots that require constant supervision, such as teleoperated robots, to robots that are able to perform their task with little or almost no human intervention.

The use of robotic systems becomes particularly beneficial when those robots require little to no supervision at all. However, such autonomous robots can be difficult to design, since they may need to cope with a large set of different situations, and not all of them can be known in advance by the designer. As such, adaptivity to unknown or changing conditions is desirable, or even a necessary property of autonomous robotic systems. In our work, robots have a high degree of autonomy, requiring no human intervention during their operation, since they adapt by themselves to unknown environments and tasks using Embodied Evolutionary Robotics (EER) approaches The online nature of EER algorithms should be stressed. Robots that must be autonomous when performing tasks in unknown and possibly dynamic environments are in strong need for online adaptation. For instance, this is the case in ocean and planetary exploration or unmanned vehicle driving applications. Therefore, in this thesis, we make very limited assumptions on the *a priori* information available to robots. This, in turn, leads to approaches that are broadly applicable, and that could be robustly implemented for a myriad of robotic tasks, using different types of robots, and requiring little expert knowledge and supervision.

1.2.1 Agents

An agent is a situated entity that perceives its environment, which corresponds to its surroundings, and acts upon it [Russell and Norvig, 2003]. In our case, we consider physically grounded (or embodied) agents, *i.e.* robots, which obtain information from the environment using sensors and apply actions using actuators, in order to fulfill its goal. The process of selecting an action based on current or previous perceptions is called *decision*. The decision system of an agent is usually termed *controller*. Figure 1.1 depicts a generic schematic diagram showing a robotic agent, its controller, and its interactions with the environment (perceptions and actions).

An agent needs to cope with the intrinsic limitations of situatedness (*e.g.* local and noisy perceptions, actions and communications, and partial observability) when making decisions in its environment. Agents can be divided into two main categories, *reactive* or *deliberative* agents, depending on whether they build representations of their environment or not. Although the line between reactive and deliberative agents can be sometimes blurry [Stone and Veloso, 2000], we deem it useful to provide a general description. Deliberative agents build and maintain such explicit representations or models of their world. These models are updated using observations and inferences performed by the agent, and are then exploited to make decisions, using reasoning, planning or prediction processes. On the other hand reactive agents, which are the type of agent used in our work, do not build models of their environment, but rather directly base their decisions on their perceptions. Purely reactive agents retrieve or compute their actions based solely on current perceptions. In that case, no internal state is maintained, and behaviors are similar

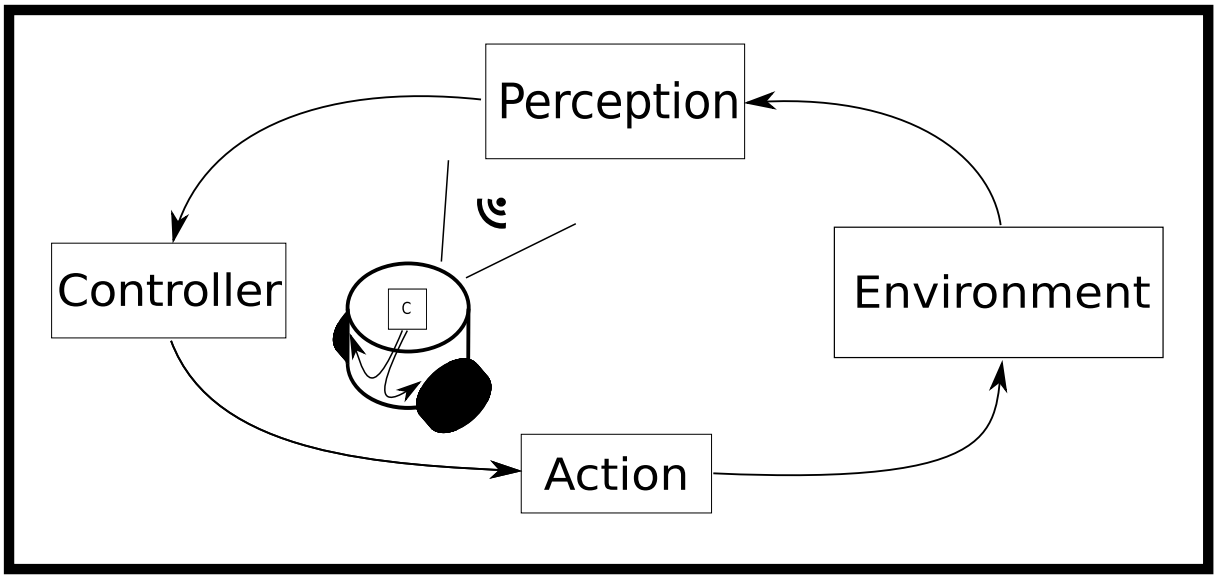

Figure 1.1 – A robotic situated agent interacting with an environment: it perceives the environment, makes a decision, and performs an action. The controller, or decision system links sensory inputs to motor outputs.

to reflexes: the agent reacts immediately to its observations from the environment, and these observations constitute the only information it can use to compute its actions. In a less restricted view, agents maintaining minimal internal state or memory of the past, can also be considered as reactive agents. Consequently, reactive agents do not use any symbolic representation or reasoning, perform planning, or maintain explicit or complex models of their environment. Reactive agents include, for example, agents with sense-and-respond behavior [Greene *et al.*, 2006], and individual agents in most Complex Systems [Holland, 1992] and in Swarm Intelligence (SI) [Bonabeau *et al.*, 1999].

Our research concerns robot swarms, *i.e.* teams of large numbers of rather simple adaptive robots (physically embodied and situated agents). In our work, each robot in the swarm is considered as a simple reactive agent that takes raw inputs from its sensors, and uses a decision system, or controller, to map sensory inputs to motor outputs. We opted for Artificial Neural Networks (ANNs) as the type of controller, which are a parametric function approximators that take inspiration from biological neural networks in animal brains and nervous systems. Neural robot controllers directly use sensory inputs, and possibly some memory of past percepts, to compute motor outputs (actions). There is a body of research concerned with Multiagent Systems (MASs), *i.e.* systems that consider the interactions of multiple autonomous agents to find solutions to given problems. These include collective robot systems, such as robot swarms. In the next section, we overview the advantages and challenges of such collective robot systems.

1.2.2 Collective Robot Systems

In this section, we describe the intrinsic advantages of collective robot systems, and list challenges when designing them. Machine Learning approaches to learn robot control can be used to overcome these challenges. These are reviewed in Chapter 2. Concretely, in this thesis, we use approaches that take inspiration from biological evolution (Evolutionary Algorithms) for distributed adaptation in swarms of robots: Embodied Evolutionary Robotics. In Chapter 3, we review work that employs

evolutionary approaches to automatically design robot control.

Advantages of Collective Robot Systems

Using multiple robots to fulfill a task has several intrinsic advantages, listed below.

Efficiency. In principle, multiple robots can perform a task much more efficiently than a single robot, since they may parallelize tasks, or divide labor by specializing in subtasks. Specialization and task-partitioning are desirable properties of multirobot systems, which may lead to dramatically decreased completion times if robots solve subtasks in parallel. A known problem in the literature of multirobot systems concerns the automatic discovery of subtasks by teams of robots, as well as how to allocate subtasks to different robots in a self-organized manner.

Fault-tolerance. If the only robot of a single-robot system breaks, the entire system stops. However, if an individual robot in a multirobot team breaks, the rest of the robots may still be able to perform the task (although less efficiently). In this sense, multirobot teams can be more robust and fault-tolerant to individual breakdowns, given that there is redundancy between the different members of the team. Consequently, teams of robots may show graceful degradation when individual failures occur, instead of leading to a complete system failure.

Cooperation. Several robots may be able to perform intrinsically collaborative tasks that single robots simply cannot. For example, a team of robots can carry cumbersome objects, that may be too heavy for a single robot. Additionally, multiple robots may be at different locations in the environment, which allows to solve collaborative tasks that may require spatial dispersion, such as deploying a mobile communication relay network.

Distributed sensing. Related to the previous point, different robots in a team may be in different areas of the environment. As such, the observations gathered by the team are more complete than the ones that a single robot would be able to collect. Additionally, availability of richer data about the environment allows to infer further information, a situation in which the whole is greater than the sum of its parts. Indeed, sensor-fusion techniques have been developed to combine data acquired by different units in the system, and to allow to integrate such data to infer further information [Fansi-Tchango *et al.*, 2014], *e.g.* to compute safe navigation paths between two distant points of the environment.

Scalability of techniques. Although scalability is not always an intrinsic property of multirobot teams, many approaches developed in the literature of multirobot and especially swarm robotic systems are scalable in terms of the number of robots. This scalability refers to efficiently increasing the productivity of the robotic system by adding more robots. Here, efficiency means limiting the increase in computational complexity to the system, due to additional robots. In that sense, it is important to note that, in swarm robotics, scalability is usually transparent, *i.e.* it suffices to add more robots to increase the productivity of the swarm. Such a transparency is accentuated when all robots are behaviorally homogeneous, because, in this case, the system is minimally disturbed when adding more robots.

Challenges of Collective Robot Systems

Despite the aforementioned advantages of robotic swarms, the design of control for such systems becomes more challenging than for a single robot. This is due to several reasons, listed below.

Increased unpredictability. Besides considering the consequences of the actions of each individual robot, the potential complex and nonlinear interactions among robots must also be taken into account. This may lead to intractable dynamics in the system. This issue is accentuated when considering parallel learning or coevolving sets of robots. In these cases, the behavior of each individual at time t results from learning based on past behaviors of other robots, instead of current ones. This increased unpredictability has been extensively studied in biological and artificial evolution, and needs to be considered when designing collective robot systems.
In our work, reactive robotic agents do not build predictive models of their environment, including other agents, but are rather constantly adapting by trial and error, which helps coping with unpredictability.

Decentralization. Generally, control needs to be applied by each robot separately, *i.e.* control is decentralized. Consequently, each robot makes decisions on the basis of local, and thus partial information. As an example, synchronization in multirobot systems is a difficult problem when there is no central entity that may access global information. An exception can be made for robot systems in which a global communication network is available, which allows for a central unit, *e.g.* a robot leader, to take decisions in a centralized manner, using information from all the robots, and transmit such decisions to the rest of the team. This, in addition to having strong requirements on the communication infrastructure, introduces both a computational bottleneck and a unique critical component in the robot leader. If the robot leader, which is a cornerstone of the system, stops working, the complete team would fail.
The approaches used in our work do not rely on any central entity or global information, thus respecting these decentralization constraints.

Global *vs.* local goals. Related to the previous point, in decentralized robotic systems, usually the local goals of each individual robot are defined to fulfill a global goal. However, the link between local and global goals is usually not straightforward. That is, defining a local objective to make an overall global goal emerge is generally challenging. This is related to the *multiagent Credit-Assignment Problem (CAP)* in learning Multiagent Systems (MAS) [Weiss, 1999]. The CAP in learning MAS relates to the problem of properly rewarding individual agents based on global performance, in a top-down manner. A possible approach is to invert the levels of goal specification, *i.e.* using a bottom-up approach, by first defining local rewards for individual goals, and then defining the global reward as their sum. However, as shown in [Hardin, 1968], agents maximizing their respective individual rewards may not lead to global reward maximization, especially in environments with limited shared resources, which is known as the *Tragedy of Commons*.
The problem of how to derive local rewards based on global goal specifications for a given problem remains an open question. In this thesis, we do not define an *a priori* global performance measure to be optimize, but rather define individual goals for the robots.

Communication needs. Managing information relaying between robots requires a communication system, which further complexifies the design of multirobot systems. Communication networks need to consider asynchronism among individuals, noisy communication channels, possible global communication requirements, and individual faults. Generally, these issues are delegated

in a network protocol. Nevertheless, when designing multirobot systems, special attention must be paid to determine what, when, how, and to whom to communicate. Further, control systems should be robust to communication problems, and costly information transfer should be limited. In our work, robots in the swarm locally exchange signals when meeting. These signals include information about their respective processes of adaptation, which is used by the robots to improve the search for adequate behaviors. Our experiments are performed in simulation, and inter-robot communication issues are thus side-stepped.

These issues need to be accounted for when designing swarms of robots. Consequently, the major challenge in these systems is how to design controllers for the robots to solve a given task. Several methodologies have been proposed over the years to engineer robot controllers [Siciliano and Khatib, 2016]. Such engineering methods are extremely time-consuming, and, since robot control is designed by hand, they are limited in the degree of complexity they can tackle, and may be error-prone. To overcome these limitations, many automatic robot control synthesis (or learning) methods have been proposed and applied, which fall in the field of Machine Learning (ML), reviewed in Chapter 2. The ML approach for robot behavior synthesis used in this thesis is based on evolutionary techniques, which are reviewed in Chapter 3. In the remainder of this introduction, we provide an outline of the thesis, highlighting the contributions of this thesis to the adaptation of swarms of robots using distributed evolutionary approaches,

1.3 Outline of this Thesis

In our work, we investigate the distributed evolution of neurocontrollers for a swarm of robots adapting online to unknown environments and tasks. These studies have led to several insights covering different topics about distributed swarm behavior adaptation. Our objective in this work is to endow swarms of robots with the ability of adapting to unknown and possibly changing environments. A family of algorithms in the field of Evolutionary Robotics, known as Embodied Evolutionary Robotics, is concerned with open-ended evolution run by each robot onboard and in an online fashion, *i.e.* once the robots are deployed for the actual task execution. As such, we have chosen the EER framework to investigate progressive online behavior adaptation for swarms of robots. Such online settings may benefit not only from efficient and effective adaptation algorithms, but also from incrementally learning appropriate behaviors for different encountered situations, which may increase the swarm adaptivity. Thus, EER approaches are potential candidates for lifelong learning, in which a robot swarm must continuously adapt to unknown and possibly changing environments over time, aiming to learn, retain, and accumulate as many skills as possible. In this thesis, we present several contributions related to such online adaptation of swarm robot behavior. The outline of the remainder of this document is as follows.

Machine Learning and Adaptation. In Chapter 2, we review work in the ML literature, with a focus on robotics, and particularly, on Reinforcement Learning (RL), which addresses behavior learning for agents interacting with an environment. We distinguish between offline and online approaches, and we situate our work in the online adaptation category. We further discuss recent ML approaches that address sets of tasks, while presenting the problem of forgetting in ANNs. Finally, we discuss our original perspective to avoid forgetting at the populational level, inspired from Dynamic Optimization approaches.

Evolutionary Robotics. In Chapter 3, we review the field of Evolutionary Robotics (ER), *i.e.* evolutionary approaches to the synthesis of robot behavior. We describe the main elements of Evolutionary Algorithms (EAs), and review existing approaches to evolve robot behavior. We then focus on evolving swarm behavior, discussing important properties in these approaches, such as the degree of homogeneity, the level of selection, and the evolution of specialized behaviors and division of labor. Finally, we present Embodied Evolutionary Robotics, the family of approaches used in this thesis, while reviewing and classifying EER work into encapsulated, distributed and hybrid approaches.

Approach for Swarm Robot Adaptation. In Chapter 4, we present the general algorithmic and experimental approach in our work. We then describe the common features of our experiments: the baseline EER algorithm (a variant of mEDEA), the robot morphology, the environments, and the simulator (Roborobo!). Additionally, we propose a set of performance measures for online evolution. These post-analysis measures integrate information over time, which provides a more reliable evaluation for online settings. We then discuss the problem of the adaptation to sequences of tasks, and how avoiding forgetting in this case can improve adaptivity in the evolutionary process. Finally, we state the research questions addressed in this thesis.

Reconsidering Selection Pressure in Distributed EER. In Chapter 5, we present our contributions on evaluating the influence of selection pressure when evolving swarm behaviors using distributed EER. In classical centralized approaches, selection pressure is usually slightly lowered to enhance diversity, which is needed to avoid premature convergence. However, distributed approaches entail different conditions for the adaptation process, which poses the question of what is the role of selection pressure in the case of distributed EER. In the chapter, we first review the notion of selection pressure in centralized and distributed approaches, as related to the exploration *vs.* exploitation dilemma. Subsequently, we discuss our methodology and experimental settings to investigate the effect of the intensity of evolutionary selection pressure at the local level using distributed EER to adapt to given tasks. In the experiments, we test different intensities of selection pressure applied on the local subpopulations in each robot, and evaluate their impact on performance. Our results show that the stronger the selection pressure at the individual level, the better the performances of the adapting robot swarm. As such, diversity, which is necessary for an effective search, seems to be naturally maintained by the local subpopulations in each robot on which selection is applied in distributed EER.

Distributed Evolution of Collaboration. In Chapter 6, we present our contributions on evolving swarm robot behavior for a collaborative task using a distributed EER approach. Learning collaborative behavior from scratch with decentralized approaches is particularly difficult, since coordination of pairs of robots needs to emerge. This is an instance of the chicken-and-egg dilemma: an individual that tries to collaborate cannot benefit from it unless other individuals are already showing collaborative behavior. In the chapter, we first review related work on the evolution of collaborative behaviors in swarms of robots, with a focus on distributed approaches. Subsequently, we present the task used in our experiments, an intrinsically collaborative item-collection task in which at least two robots are simultaneously needed to collect items. Furthermore, to emphasize the need for coordination, we complexify the problem by considering different subtasks (types of items), which require robots to choose particular actions for the subtasks to be solved. as well as our methodology and experimental settings. Our results show that a swarm of robots using a distributed EER approach can learn such a collaborative task. Additionally, we show

that, without any specific mechanism to enforce task distribution over the robots, the algorithm allows to learn collaborative behaviors that do not neglect any subtask, *i.e.* items of all colors are collected.

Distributed Evolution of Neural Topologies. In Chapter 7, we present our contributions to the distributed evolution of the topology and the weights of swarm robot neurocontrollers. Evolving the topology of neurocontrollers allows for increasing their expressivity, by adding additional structure, such as neurons and connections. Effective algorithms to evolve the topology of ANNs require marking and keeping track of the order of added neurons and connections. In centralized settings, this is straightforward, since global information is available. However, in distributed settings, this may be challenging. In the chapter, we review work in topology evolution, in both centralized and distributed settings. Subsequently, we present Gene Clocks, our proposed decentralized mechanism, which takes inspiration from Distributed Computer Systems to mark neural innovations without resorting to global information when evolving neural topologies in distributed EER. In our experiments, we compare two marking mechanisms, one that uses global information, which should not be available to robots, and our proposed approach, which exclusively relies on local information. Our results show that our proposed method does not lead to a loss of performance, nor a quantitative difference in terms of size of the evolved ANNs, while providing a truly decentralized mechanism that depends only on local information.

Avoiding Forgetting at the Populational Level. In Chapter 8, we present our contributions to the evolutionary incremental adaptation to sequences of tasks. We describe our proposed algorithm, Forgetting-Avoidance Evolutionary Algorithm, which maintains a trace of phylogenetic information, the fitness values of the ancestors of the individuals in the population, to promote the retention of old skills. The lineages of each genome in the current population are stored and exploited during the reproduction in the EA, to slightly bias selection and variation toward individuals whose ancestors performed well, *e.g.* in a previous task. We present a set of preliminary experiments to test the algorithm, in a centralized version, and we analyze the obtained results. We conclude that our algorithm allows for a slight improvement in readaptation to conditions seen in the past, and it limits the impact of forgetting. Our results, although preliminary, show that the approach of avoiding forgetting at the populational level by exploiting lineage information is promising, and further research is needed to ascertain the impact of different components of the algorithm. In our experiments, we used a centralized algorithm, which is not suited for decentralized swarms of robots. Consequently, we discuss how this algorithm could be adapted to distributed settings in a swarm of evolving robots, since it does not rely on global information.

Conclusion and Perspectives. In Chapter 9, we provide a summary of this thesis, with a focus on our contributions. We then discuss future work and possible perspectives of extension and further investigation of the research questions presented in this manuscript related to the distributed adaptation of behaviors in swarms of robotic agents.

Part I

Methodological Background

2

Machine Learning and Adaptation

Contents

In this chapter, we present an overview of Machine Learning (ML) from the perspective of multiple adaptive robots. As it is known from Nature, living organisms are able to adapt from previous experience and improve their skills on tasks with practice. Such tasks span from sensorimotor skills, such as locomotion or grasping, to more cognitive tasks, such as classification, action planning, or communication. In our work, we investigate learning and online adaptation of robot swarms to given environments and tasks. On the one hand, this includes ML approaches to multirobot control learning. On the other hand, mechanisms that allow for robots to adapt during operation need to be considered.

The field of ML is concerned with the design of computational systems that learn, where the major question could be formulated as: "*how to design computer systems that automatically improve their skills to solve given problems with experience*" [Mitchell, 1997]. In our case, these systems are robots, that need to learn policies, *i.e.* decision functions, to solve tasks in their environment. ML approaches to learning such policies fall in the field of Reinforcement Learning (RL). In our work, we use a type of RL algorithms called direct policy search. These algorithms focus on optimizing parametrized policies with respect to a single aggregate performance measure. While such a measure provides few information to guide the search for policies, which is challenging, it has limited data or feedback requirements, which broadens the applicability of policy search. Policy search algorithms typically aim to optimize single policy for an agent with respect to a fixed task.

On a different axis, adaptive robot swarms face unknown and possibly changing environments. Consequently, they are in need of mechanisms that gradually change such policies to fit their current conditions. In online ML approaches, a learning system is modified over time to adapt to new data or environmental feedback. Lifelong Machine Learning (LML) considers an online learning approach, where the system faces and progressively adapts to new learning problems while retaining solutions to previously learned ones. In the case of lifelong adaptive swarms that gradually learn policies, the focus shifts from learning a single policy to learning sets of policies: at any moment, each agent in the swarm runs a policy, and, when adapting, the agents in the swarm learn different policies over time. Consequently, for a swarm of robots to adapt over time to unknown and possibly changing conditions, they need mechanisms that provide adequate adaptivity. In our work, we use Evolutionary Robotics (ER) approaches, which can be seen as one type of policy search RL to learn robot behavior, taking inspiration from natural evolution. These approaches rely on populational optimization algorithms, *i.e.* Evolutionary Algorithms (EAs), to progressively optimize robot control. In Chapter 3, we review ER, with a focus on learning swarm robot control.

An interesting link can be made between our problem of adaptation of swarm robot behavior and Dynamic Optimization Problems (DOPs), specifically regarding approaches in Evolutionary Dynamic Optimization (EDO) , *i.e.* using EAs for DOPs. In DOPs, the goal is to accurately and rapidly track a moving optimum in the search space, or a set of moving optima. Some approaches in EDO rely on maintaining diversity in the population of candidate solutions that track the optima over time. Further, some of these approaches maintain some individuals in the population that performed well in the past. This aims to allow for efficient readaptation when changes occur, especially if those changes are cyclic, *i.e.* if previous conditions reappear. In our work, environmental or task changes may indeed be cyclic: once it adapted to a certain task, and then to a new task, a robot swarm may benefit from retaining controllers that solved the first task, should it reappear. In this sense, the swarm could readapt to a previously learned task much more efficiently if the swarm retains the behaviors that solved the first task in the population of its EA.

In the remainder of this chapter, we first provide an overview of different types of ML systems, depending on their respective data requirements, with a focus on RL for robot policies, and specifically policy search algorithms. Subsequently, we discuss the difference between offline behavior optimization and online behavior adaptation, as is investigated in this thesis. We then discuss Lifelong Machine Learning (LML), which targets efficient and effective learning systems that adapt to sequences of tasks or environments. We focus on approaches using neural networks, the controller representation models that we use in our work. We describe the problem of forgetting in ANNs, and existing approaches aiming to alleviate this problem. Finally, we discuss approaches in EDO that rely on maintaining previous individuals in the population to better track moving optima, while discussing how could adaptive swarms of robots benefit from such approaches to better adapt to changes.

2.1 Machine Learning for Robot Control

Machine Learning (ML) is a scientific field concerned with building artificial systems that improve their performance over time, based on training examples or past experience [Alpaydin, 2010]. Otherwise stated, the goal of any ML system is to learn how to solve a problem given some data. Mitchell provided a general definition as "*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T,*

as measured by P, improves with experience E." [Mitchell, 1997]. Different kinds of ML techniques have been applied to automatically build robot behavior.

Families of ML techniques are usually divided in categories, depending on one of two criteria: type of data or experience exploited when learning, and type of problem addressed by the technique. These two points of view are somewhat overlapping, since addressing a certain kind of learning problem may require a specific type of data. ML families regarding the type of data or feedback available to the learning system include supervised, unsupervised, semi-supervised, and reinforcement learning.

Supervised Learning

Supervised Learning (SL) [Hastie *et al.*, 2009, Chapter 2] exploits *labeled* data examples for the system to learn. SL approaches use a training base consisting of a set of data examples along with labels that indicate the expected system output, or ground truth. Each label corresponds to the output that the optimal system should provide when fed a given example as input. These approaches are used in classification, prediction and regression problems. SL techniques include Artificial Neural Networks with the backpropagation algorithm [Rumelhart *et al.*, 1986], decision trees [Rokach and Maimon, 2005, Chapter 9], learning classifier systems [Lanzi *et al.*, 2003], and Gaussian Processes [Rasmussen and Williams, 2006], among others.

One of the main issues with SL approaches in robotics is that they are very data-demanding. Consequently, complete end-to-end robot control systems are not usually learned using SL techniques, because it is not generally possible to provide data examples of the right action to perform in every situation. A promising research direction in ML for robotics consists in developing data-efficient algorithms that have fewer data requirements, and maximize the use of the limited amount of available data [Deisenroth *et al.*, 2015, Mouret, 2016]. Some ML approaches try to limit such data requirements, for example by using unlabeled data, which is less expensive to acquire.

Unsupervised Learning

Unsupervised Learning (UL) [Hastie *et al.*, 2009, Chapter 14] exploits unlabeled data examples for the system to learn hidden patterns or intrinsic structure of the data. UL techniques use a training base consisting of a set of *unlabeled* data examples, representative of the distribution of inputs of the problem. For instance, these data examples may be images or robot motion patterns that the system needs to categorize. UL techniques typically address clustering, dimensionality reduction, and density estimation problems. UL approaches include k-means [Lloyd, 1982], self-organizing maps [Kohonen, 1998], and density approaches [Ester *et al.*, 1996], among others.

Unlabeled data exploited by UL algorithms is relatively inexpensive to acquire. However, it is also less informative than labeled data, which limits the possibilities of UL approaches. Therefore, semi-supervised learning has been proposed to increase such possibilities, which relies on enhancing a large unlabeled dataset with a smaller set of labeled data.

Semi-Supervised Learning

Semi-Supervised Learning (semi-SL) [Chapelle *et al.*, 2010] exploits both a large set of *unlabeled* data examples, and a smaller set of *labeled* data for the system to learn, thus falling in between SL and UL. Labeling data for SL may be very costly, and it may be unfeasible to obtain a fully labeled dataset. However, acquiring unlabeled data is relatively inexpensive. As such, there has been effort in developing semi-SL techniques that can exploit both a small set of labeled data, and a larger set of unlabeled data. Typically, semi-supervised techniques rely on the assumption that

data points that are close to each other are likely to share a label. As such, they may use Bayesian learning to automatically infer labels, based on similarities with respect to labeled data [Cozman *et al.*, 2003]. Such techniques address similar problems to those in SL, *i.e.* classification, prediction, and regression.

Applications of the aforementioned families of ML approaches (SL, UL, semi-SL) in robotics are mainly found in submodules of robot control systems. In the next section, we introduce Reinforcement Learning (RL), which is a family of methods that learn end-to-end control functions, by addressing their design as sequential decision-making problems. We provide a classification of RL subfamilies, while listing some of its applications in robotics. Finally, we situate the evolutionary approaches used in our work within the class of direct policy search RL algorithms. Evolutionary approaches to learn robot control are further detailed in Chapter 3.

2.1.1 Reinforcement Learning

RL approaches [Sutton and Barto, 1998] allow a robot to autonomously discover appropriate behaviors through a trial-and-error process of interactions with the environment. These approaches focus on learning decision or control functions, which are called policies, for agents that interact with an environment, and receive generally sparse reward or punishment feedback signals as data to guide learning. Such techniques focus on sequential decision-making problems, or control problems over time, by learning a policy mapping the observations of the agent to the action to choose.

Consequently, RL approaches provide a set of algorithmic tools to automatically build behaviors for complex problems, such as those found in robotics [Deisenroth *et al.*, 2013]. On the one hand, they may first learn low-level control primitives, and then higher level modules based on such primitives. Alternatively, they may directly build complete policies as monolithic architectures that map sensors to motors. This, in turn, eases the engineering of such behaviors, by limiting the amount of effort to be done by the human designer.

Generally, RL approaches rely on a sequence of common phases: exploring the policy space around the current policy to avoid getting stuck in local optima, evaluating the policy with respect to its expected long-term reward, and improving the current policy using this evaluation [Sutton and Barto, 1998]. Usually, the part of an RL algorithm that estimates the value of a policy is called the *critic*, and the part that decides the actions to make depending on the state is called the *actor*.

In this sense, RL approaches can be classified in a continuum between pure critic methods and pure actor methods. Such approaches make different assumptions, and require different *a priori* information. Here, we provide a classification of RL algorithms in this range.

Critic-based

Critic-based algorithms focus on learning a value function, which links a state, or a state and an action, to the long-term expected reward for the policy. Using environmental reward feedback, these methods try to learn the optimal value function to compute the expected reward that policies may obtain over time. Consequently, they are also known as value-based approaches. Such a value function is then used to avoid querying the environment to evaluate a policy, *i.e.* the value function is used as a surrogate model of the long-term reward, so the evaluation of the policy can be estimated while avoiding costly executions in the environment. In purely critic-based approaches, once the value function is learned, a policy consists in greedy, or myopic, selection of

the action that maximizes value given a current state.

The main drawbacks of critic-based RL algorithms is that (a) they cannot be applied in continuous or high-dimensional state and action spaces, which is the case in robotic applications; and (b) the greedy approach to generate a policy can be very sensitive to uncertainties, since small errors propagate through time and can become considerably high. Consequently, greedy strategies may be highly inefficient, especially when the estimated value function is not optimal. On the other hand, the advantages of critic-based algorithms, as stated above, are that (a) they help reduce the number of policy evaluations in the actual environment, especially expensive in robot applications; and (b) there exist theoretical guarantees of global optimality for certain critic-based algorithms.

Critic-based RL approaches include: Value-Iteration [Sutton and Barto, 1998, Chapter 4], and Temporal Difference [Sutton and Barto, 1990] algorithms, which learn a value function that, given a policy and a state, returns the expected long-term reward for the agent; and Q-learning [Watkins, 1989] and SARSA [Rummery and Niranjan, 1994], which learn a value function that, given a policy, a state and an action, returns the expected long-term reward for the agent. Applications of purely critic-based RL algorithms in robotics are scarce, due to intrinsic complexity limitations of these methods in continuous environments. Among existing applications, we find locomotion and obstacle-avoidance for autonomous vehicles [Touzet, 1997] and bipedal robots [Morimoto and Atkeson, 2003, Hester *et al.*, 2010], and manipulation with robotic arms [Yamaguchi and Atkeson, 2016].

Actor-based

Actor-based algorithms focus on improving policies over time. They progressively improve an initial policy, based on reward feedback from the environment. Since these algorithms focus on finding a policy for the task at hand, they are also known as policy search methods [Deisenroth *et al.*, 2013]. Actor-based methods typically use a parametrized policy representation, that provides the action to apply depending on the current perceived state. As such, the problem of learning a policy becomes the problem of finding the policy parameters that maximize the long-term reward for the agent. In our work, parametrized policies correspond to neural network robot controllers that map sensors to motors.

The main drawbacks of actor-based approaches are that (a) the variance of the long-term reward in considered policies may be high; and (b) execution of poor policies may be dangerous. To reduce the variance, each policy should be evaluated a considerable number of times, which is very expensive. Additionally, since very poor policies may be evaluated, even in late stages of learning, this can lead to problematic situations in which the robot is endangered or damaged, and it can hurt humans or structures in the environment. On the other hand, the advantages of these approaches are that (a) policy search approaches can learn directly in continuous state and action spaces, if given a proper continuous representation for the policy, and (b) assumptions on available data may be weaker, especially in *gradient-free* approaches. Continuous spaces are especially desirable in robotics, since sensor and motor signals are typically continuous. Additionally, in purely actor-based approaches, policy evaluation is reduced to executing a policy in the environment and measuring a quality indicator, *i.e.* its performance or fitness value, thus requiring fewer information from the environment (*cf. gradient-free* methods below). There are two actor-based approaches depending on the availability of gradients on the long-term reward when evaluating a policy: gradient-aware and gradient-free.

Gradient-aware. These methods exploit information on the gradient of the cumulated reward of a policy with respect to variations in the policy parameters to progressively improve such parameters. Gradient-aware RL approaches include REINFORCE algorithms [Williams, 1992] and Policy Gradients with Parameter-based Exploration (PGPE) [Sehnke *et al.*, 2008, Rückstiess *et al.*, 2010]. Examples of robotic applications of gradient-aware methods are unicycle riding [Deisenroth and Rasmussen, 2011], table-tennis playing [Peters *et al.*, 2010], and humanoid reaching tasks [Tangkaratt *et al.*, 2014].

Gradient-free. These methods do not require explicit information on the gradient and do not need to estimate it, but rather rely only on an evaluation of the overall performance of the policies. Learning is done by solving a black-box optimization problem [Hansen *et al.*, 2010]. There exist different approaches to solve such problems. These include local search (hill-climbing [Russell and Norvig, 2003], tabu search [Glover, 1989], simulated annealing [Kirkpatrick *et al.*, 1983], ...); and population-based stochastic metaheuristics, (Evolutionary Algorithms (EAs) [Eiben and Smith, 2003], Particle Swarm Optimization (PSO) [Kennedy, 2011], Ant Colony Optimization (ACO) [Dorigo *et al.*, 2006], ...). These metaheuristics usually take inspiration from nature to define an "intelligent" way of searching a space to optimize solutions [Yang, 2008]. In our case, the solutions to be optimized are policies, or control functions, for a robotic agent to solve a task. Examples of robotic applications using gradient-free actor-based approaches include gait learning for locomotion in legged robots [Yosinski *et al.*, 2011], ball-collecting and box-pushing tasks [Zimmer and Doncieux, 2017], and robotic manipulator reaching tasks [Chatzilygeroudis *et al.*, 2017]. The learning algorithms used in this thesis belong to the category of gradient-free policy search algorithms, and more specifically, to the family of EAs, detailed in Chapter 3.

Actor-Critic

Intermediate approaches, referred to as actor-critic algorithms, [Sutton *et al.*, 2000, Konda and Tsitsiklis, 2000] aim at combining features from both the critic-based and the actor-based families. Actor-critic algorithms alternate between two phases. First, a critic, or current approximation to the value function, is computed, based on available data; then an actor, or explicit representation of a policy, is updated based on the values returned by the critic. After that, the policy is run, which in turn provides new data to improve the critic, and the process iterates. A certain random noise is usually added to policies to ensure exploration, and avoid premature convergence to local optima. The advantage of actor-critic approaches is that they combine good features of value-based approaches (*e.g.* building and exploiting a value function) and of policy search approaches (*e.g.* coping with continuous action spaces). Consequently, numerous examples of applications of RL in robotics fall in the class of actor-critic approaches. These include, for example, helicopter flight [Kim *et al.*, 2004], robotic arm manipulation [Kober and Peters, 2009], and locomotion with multi-legged robots [Gu *et al.*, 2016].

RL for Multiple Robots

Regarding multirobot teams and swarms, learning approaches are less common than in the single-robot case. Most techniques fall in one of two general approaches. On the one hand, approaches based on solving Decentralized Partially-Observable Markov Decision Processes (Dec-POMDPs) [Oliehoek and Amato, 2016] learn a decentralized joint policy for a multirobot team. This is typically done by using critic-based approaches that compute a value function for the team of robots, while addressing the Credit-Assignment Problem (CAP), *i.e.* crediting in an adequate

manner each robot in the team for their respective contributions to the goal. Solving Dec-POMDPs in an exact manner is highly expensive. As such, approximative algorithms have been proposed to address realistic applications [Dibangoye *et al.*, 2009]. Examples of application of Dec-POMDPs in robotics include multirobot box-pushing and package delivery [Amato *et al.*, 2015], multirobot aggregation [Eker *et al.*, 2011], and cooperative bartender-and-waiter tasks [Amato *et al.*, 2016].

On the other hand, gradient-free policy search methods, especially with population-based metaheuristics such as EAs or PSO, have also been used to learn multirobot and swarm robot behavior. These approaches generally find a set of high performing policies using approximative approaches, and are less computationally expensive than Dec-POMDPs. Examples of such applications of PSO include navigation with obstacle avoidance [Pugh and Martinoli, 2006, Di Mario *et al.*, 2013], exploration of unknown environments [Wang *et al.*, 2016], and search-and-rescue applications [Couceiro *et al.*, 2013]. As said before, in our work we use Evolutionary Algorithms (EAs) to learn swarm robot behavior, which are direct policy search approaches. In our case, EAs may be seen as black-box optimization methods to learn swarm robot control. Evolutionary Robotics (ER), *i.e.* using EAs to learn robot behavior, is discussed in Chapter 3. In that chapter, we further develop our discussion on evolutionary approaches to learning swarm robot control, particularly on EER methods, the specific class of EAs used in our work for decentralized swarm robot behavior adaptation.

2.2 Offline Learning *vs.* Online Adaptation

When developing ML algorithms for behavior learning in a robot swarm, an important question is: when does the learning algorithm act? Regarding this question, there exist two main categories of algorithms: offline and online.

2.2.1 Offline Learning

Offline learning algorithms consider two separate stages for the ML system: the learning phase, in which the entire dataset is available and used for learning; and the exploitation phase, in which learning stops, the learned model is fixed and used for the goal task. In offline learning algorithms, the dataset is available beforehand, which allows for more flexibility in the design of the algorithm. Data can be either passed to the system at once, or it could be divided in different subsets. On the other hand, offline learning algorithms do not adapt the learned system if conditions change. In such situations, they rather rely on learning robust solutions to the problem that can cope with changes.

In most of the existing studies in learning swarm robot behavior, ML algorithms are used as offline optimizers to find effective robot behaviors. During the learning process, many behaviors are tested and modified. When the termination condition is met, the process stops and it returns a controller, usually the best one ever found. At this point, the resulting controller is fixed and is deployed in the swarm. Consequently, the algorithm is responsible for building a performing behavior before starting to actually exploit it in the real task. Subsequently, the behavior is no longer modified once the robots have been deployed for actual operation. Offline methods rely on an optimization process on an environment that has to be defined *a priori*. In some cases, this may not be possible, since part of the environmental features may be unknown. Additionally, if the conditions change over time, behaviors that have been learned offline, in a different environment or task, may not be fit after the change.

2.2.2 Online Adaptation

Online learning algorithms consider a single learning phase. In this phase, the algorithm receives the training data in a sequential manner, and it must progressively adapt as the training data becomes available. During this process of progressive adaptation, the learning system is simultaneously exploited. Consequently, fast and effective adaptation is a strong requirement: the system is operating for the actual task while learning, so the performance on every trial counts. Since online learning algorithms do not have the entire dataset at their disposal, the process of online learning is more challenging. Online algorithms need to be able to overcome partial knowledge of the dataset to learn the task. On the other hand, these algorithms are potentially able to adapt to tasks that vary over time, and thus to non-stationary learning problems. A straightforward approach to online adaptation consists in running an offline learning algorithm from scratch after new data is received. However, this is extremely inefficient, since new data does not necessarily mean that the task has changed, and learning new tasks from scratch is expensive. Efficient online approaches take advantage from previously acquired knowledge, and provide flexible mechanisms for gradual adaptation.

In the case of approaches for online swarm robot adaptation, behaviors are learned while executing the actual task. As such, constraints are different with respect to the offline case: previous data is not directly available, and adaptation never stops. Precisely, since online learning is open-ended (in the sense that adaptation never stops), such approaches present themselves as good candidates to progressively adapt control systems to unknown and changing environments. Online adaptation of behaviors poses additional challenges with respect to offline optimization, since learning while operating requires an efficient system that is able to recognize new situations, and properly modify the behaviors in a timely manner, while maintaining high performance on the system. Additionally, in online settings, robots may be confronted with dangerous environmental conditions, and evaluated behaviors could also endanger them. This means that special attention should be paid to rapidly analyze such dangerous situations and avoid them.

In this thesis, we investigate how to promote adaptivity in a swarm of robotic agents that learn while operating, *i.e.* that perform online adaptation. To design highly adaptive swarm systems, one possibility would be to provide them with the ability to incrementally adapt to sets of tasks. In the next section, we discuss Lifelong Learning approaches for such an incremental adaptation.

2.3 Adaptation to Several Tasks

For teams of robots to be truly autonomous, they must be able to adapt to multiple unforeseen and changing situations, and to retain knowledge from previous problems. Otherwise stated, adaptation to changing conditions requires learning multiple tasks. Ideally, adaptation is open-ended, *i.e.* learned behaviors are subject to revisions over time, and they are progressively adapted if needed in a never-ending process. Therefore, online behavior learning algorithms should allow for accumulating skills over extensive periods of time, and then knowing when to apply them. However, in real-world situations, different problems are not simultaneously presented, and learning mechanisms have to be adapted to the sequential nature of such open-ended scenarios. Specifically, algorithms for swarm behavior learning should allow for incrementally learning different behaviors when facing a sequence of different tasks.

On the one hand, Embodied Evolutionary Robotics (EER) algorithms have been proposed to deal with multirobot distributed learning in online contexts, but their potential to accumulate skills over time has not been tested yet. On the other hand, several ML algorithms have been shown

to be able to learn multiple skills. However, they have been only been applied for classification and regression tasks with SL techniques, and not to robot control. Dynamic optimization techniques [Kamien and Schwartz, 2012], have also been proposed to deal with optimization problems that change over time. However, in that case the focus was set on accurately tracking a moving optimum in the search space, rather that in retention and cumulation of previous optima seen as different skills.

Learning several tasks comes in different, but related flavors covered by several paradigms in the ML community. In these approaches, a set of tasks is learned by a system, either over time or simultaneously. As skills are learned by the corresponding system, *e.g.* a robot controller, special mechanisms are applied to take advantage from learning and share knowledge between tasks. The differences between the types of approaches concerns the goal of learning with respect to the tasks, and when is learning performed. Such techniques have been mainly applied to classification and regression tasks, rather than to robot control. Here, we classify ML approaches with several tasks with respect to their assumptions, goals, and properties, into Multi-Task, Transfer, and Lifelong Learning, following [Chen *et al.*, 2016].

Multi-Task Learning

Multi-Task Learning (MTL) [Caruana, 1997] considers learning a set of tasks simultaneously. Such approaches jointly optimize the complete set of tasks by exploiting similarities between the tasks. These similarities allow for improved learning, as compared to learning each task in isolation, by exploiting knowledge on possibly related tasks. The tasks are learned in parallel by building a model using multiple datasets from different tasks. The main assumption is that what is learned for each task can be useful for other tasks. Consequently, MTL exploit features that are built for each task to better learn the others. This can be seen as a parallel inductive transfer mechanism, *i.e.* inductive bias is provided to the tasks being learned in parallel, from what is already known from previous learning on such tasks.

Different approaches exist in MTL. *Representational sharing* approaches push toward representations that are useful for more than one task [Caruana, 1993]. In [Caruana, 1994], the authors identify mechanisms that help improving performance in MTL: building shared representations, such as hidden layers in ANNs, which allow for sharing precomputations among different tasks; extracting inputs relevant to multiple tasks, which allows for shared feature selection between the tasks; or automatically discovering relationships between tasks, which allows for adequate guidance on which tasks should share information. In [Silver *et al.*, 2008], the authors investigate not only sharing internal representations among tasks, but also output modules. They add a set of *context* inputs to the MTL system to provide information on which task is currently addressed. On the other hand, *functional sharing* approaches to MTL use implicit soft sharing mechanisms based on distances between task representations to learn multiple tasks in parallel. For example, in [Evgeniou and Pontil, 2004,Liu *et al.*, 2013], regularization mechanisms based on such intertask distances are used to induce learning that it useful for all tasks.

Transfer Learning

While MTL focus on learning several tasks simultaneously, in some cases this is an unrealistic assumption. For instance, data from different tasks may not be simultaneously available, thus requiring learning methods that can be sequentially applied, in an online manner, *i.e.* when tasks become available. Transfer Learning (TL) [Torrey and Shavlik, 2009, Pan and Yang, 2010] learns a set of *source* tasks, which is subsequently exploited when learning a set of *target* tasks. Here,

the focus is set on improving learning on the target tasks by leveraging information from previous learning on source tasks. As such, TL approaches are not concerned with retaining knowledge on the previously learned source tasks, but rather with exploiting knowledge obtained in these tasks as stepping stones to subsequently learn the target tasks more rapidly and efficiently. TL has two main differences with respect to MTL: in TL, target tasks are not available when learning source tasks, but they rather become available after source tasks are learned; and in TL, the goal is limited to learn target tasks, and source tasks are used to improve such a learning, without requiring to retain them afterwards.

In TL, different approaches exist to exploit knowledge on source tasks to better learn target tasks, depending on what is transferred [Pan and Yang, 2010]. In *instance transfer*, data examples from source tasks are used when learning target tasks, which allows for an improved performance and speed of learning in target tasks [Smith *et al.*, 1997, Jiang and Zhai, 2007]. Since blindly transferring data examples may be harmful for target tasks, the importance of data examples from source tasks may be weighted by an adaptive parameter. In *feature-representation transfer*, the system learned using the source tasks is directly transferred as a prior to learn the target task [Raina *et al.*, 2006, Argyriou *et al.*, 2007]. The idea behind this approach is that learning an adequate feature representation using related source tasks may help learning target tasks. *Parametric transfer* assumes that source and target tasks have similar requirements in terms of parameters or hyperparameters (such as learning rates) [Lawrence and Platt, 2004, Evgeniou and Pontil, 2004]. Thus, appropriate parameters found in the source tasks are reused in the target tasks to improve learning [Torrey *et al.*, 2008]. *Relational-knowledge transfer* considers known relationships between source and target tasks in relational domains, for example in communication or social network data applications. The rationale behind these approaches is that building mappings between source and target tasks allows for extrapolating known relations from source learned models to target domains. While most TL techniques have been applied in supervised and unsupervised context, there exist some work on TL for Reinforcement Learning problems. In [Taylor and Stone, 2009, Taylor and Stone, 2011], the authors provide an in-depth survey of TL in RL.

2.3.1 Lifelong Machine Learning

As said before, TL focus is set on learning target tasks, while the source tasks are seen as stepping stones to ease such a learning. On the other hand, LML [Silver *et al.*, 2013] approaches focus on incrementally learn sequences of tasks, while retaining previous tasks and exploiting (transferring) previous knowledge to new tasks as they are progressively presented to the system. The training data is provided to the system in a sequential order (as in single-task online ML), and the goal is to learn, retain and accumulate knowledge on multiple tasks while leveraging previous learning (as in MTL). As such, LML may be considered as incremental online MTL [Chen *et al.*, 2016], since the goal is to learn and share knowledge among a set of tasks, while addressing them online, *i.e.* when they become available to the system. A more precise definition, from [Silver *et al.*, 2013], states:

"*Lifelong Machine Learning, or LML, considers systems that can learn many tasks over a lifetime from one or more domains. They efficiently and effectively retain the knowledge they have learned and use that knowledge to more efficiently and effectively learn new tasks.*"

Otherwise stated, LML focus on continuous learning of tasks in sequence. The corresponding system learns a first task, exploits (transfers) this knowledge when learning a second one, and incrementally continues to learn new tasks without losing previous ones. The central questions to develop LML algorithms are (a) how to transfer knowledge from previously acquired tasks,

and (b) how to retain previously acquired knowledge (or how to avoid forgetting). In this sense, LML is concerned with a continuous process of learning, where knowledge from previous tasks is consolidated and transferred to improve learning of new tasks. Other names for these approaches are learning to learn [Thrun and Pratt, 1998], or never-ending learning [Mitchell *et al.*, 2015]. In LML, different approaches exist to help progressively retaining and exploiting previous learning in an open-ended manner.

In explanation-based ANNs [Mitchell and Thrun, 1993, Thrun and Mitchell, 1995], backpropagation is used to learn the parameters of an ANN robot controller that runs an RL algorithm. The algorithm progressively builds task-independent transition models over time, based on previous evaluations. The partial derivatives on the value function, computed for the backpropagation algorithm, are extracted and stored. Both transition models and the partial derivatives are then transferred when learning new tasks to facilitate their learning.

In LML based on MTL [Silver and McCracken, 2003, Silver and Poirier, 2004, Silver, 2013], MTL systems with representational sharing (*cf.* above) are adapted for sequential learning, instead of simultaneous learning. In their work, the authors focus on techniques to sequentially consolidate and retain previous knowledge to avoid erasing it. These techniques include using a long-term ANN that retains knowledge from previous tasks, and rehearsal mechanisms that retrain the learning system using previous data. Knowledge transfer is intrinsically done by shared representations, as in MTL. In [Poirier and Silver, 2005], the authors investigate the effect of task order on the retention of previous knowledge, concluding that such an order especially influences the performance of the system during the first tasks.

In the Efficient Lifelong Learning Algorithm (ELLA) [Ruvolo and Eaton, 2013b], a sparsely shared basis representation for all tasks is maintained, so new tasks can exploit it. Knowledge is transferred to new tasks through this shared representation, which is explicitly retained, and progressively refined over time to maximize performance over all tasks. The sparseness of the representation helps avoiding negative interference between unrelated tasks. The authors show that ELLA yields similar performance as compared to simultaneous learning of the entire set of tasks, with a dramatic increase in learning speed, over 1000 times faster. In [Ruvolo and Eaton, 2013a], the authors extend the algorithm to actively select task order over a subset of currently available tasks so as to increase performance on future tasks. In [Ammar *et al.*, 2014], the authors extend ELLA to learn policies for sequential decision-making problems using policy gradient RL methods. Their approach has robust theoretical guarantees, and they experimentally show a large increase in learning speed.

It is worth mentioning the work by Silver *et al.* [Silver *et al.*, 2013], a survey and position paper where the authors review previous work on LML, and argue for a systems approach to LML and ML in general. The authors describe effective (performing) and efficient (fast) knowledge retention and knowledge transfer as the goal of LML systems. They further sketch in general terms the essential components, and a reference framework for LML systems: universal and domain knowledge bases, as well as an inductive learning system that extract relevant knowledge from the bases. Such a system can be then coupled to a ML system that exploits available knowledge, and that provides feedback for adequate retention and consolidation when populating the universal and domain knowledge bases.

In the next section, we discuss the problem of forgetting when applying incremental lifelong adaptation to sequences of tasks in ANNs. This is directly linked to our work on swarm of robots that collectively adapt in an continuous manner to possibly changing environments or tasks.

Learning in Sequence: Catastrophic Forgetting in ANNs

Our goal is to allow swarms of robots to adapt to possibly changing environments, which relates to LML. Indeed, promoting retention of past tasks, *i.e.* avoiding forgetting, as in LML could be beneficial for the incremental adaptation to different environments or tasks that are presented in sequence. When isolated ML algorithms are presented with new tasks, previous knowledge may be erased, a problem that is known as *catastrophic interference* or *catastrophic forgetting*. This issue is accentuated when using ANNs as representational structure for learning, due to the fact that ANNs use a distributed representation to store learned information. In such distributed representations, new knowledge is not stored in a single *locus* in the structure of the ANN, but rather stored by the interaction of a subset of the synaptic connections. Consequently, when a new task is presented to an ANN, all the weights are updated, which heavily disrupts past knowledge, thus leading to almost instantaneously erasing what was previously learned.

This forgetting problem in ANNs has been studied in Supervised Learning with a focus on retaining learned information when training with new data [French, 1999]. Some approaches aiming to alleviate this issue have been proposed. For example, semi-distributed representation architectures have been studied to reduce the overlap of tasks in the structure of the neural networks [French, 1992]. These approaches try to store knowledge of new tasks in different parts of the ANN to avoid erasing old tasks, which may be seen as promoting modularity in the neural structure. Other approaches to reducing the impact of forgetting concern the rehearsal or retraining of old tasks to help retain them, by adding data examples of such tasks among the examples of the current task. However, rehearsal approaches are limited, since previous data examples may be costly to maintain, or they may even not be available anymore. An approach that sidesteps such limitations, called pseudorehearsal, does not require access to the actual examples of previous tasks [Robins, 1995]. In pseudorehearsal, when a task is finished, a learned model of the task is stored, which then serves to generate examples for retraining on this task.

In the next section, we present a different perspective to avoiding forgetting, taking inspiration from population-based approaches for Dynamic Optimization Problems (DOPs).

2.3.2 Dynamic Optimization Perspective

DOPs are problems where the goal is not only to find the optimum of a given function, but also to track it over time as the problem changes. In DOPs, the focus is not set on retaining old optima, but on efficiently and rapidly adapting when changes occur. As such, the problem does not really concern learning or retaining several tasks. However, when using evolutionary approaches for DOPs, which is known as Evolutionary Dynamic Optimization (EDO) [Nguyen *et al.*, 2012], several algorithms have been proposed that maintain in the population some candidate solutions that were good in the past. The goal of such approaches is to maintain a high level of diversity, with promising solutions gathered in the past, to rapidly and effectively readapt to environmental changes, especially to cyclic ones. In our work, we take a similar point of view, but our focus is set on limiting task forgetting by promoting the retention of solutions that were good in the past. Such a focus is somehow different to usual approaches for DOPs, which rather focus on rapidly adapting to environmental changes.

Concretely, the proposed approach in Chapter 8 tries to maintain individuals with high performance in previous tasks in the population of the EA, while adapting to new tasks. As such, the forgetting problem is reduced at the level of the population: distinct subpopulations focus on either optimizing performance on the current task, or retaining previous tasks, with knowledge on the different tasks spread through the global population. By defining the problem of forgetting

at the populational level, our vision builds a bridge between the problem of task retention in ML, and populational approaches in EDO to cope with task changes.

2.4 Conclusion

In this chapter, we provided an overview of ML approaches, especially actor-based policy search RL in robotics. We then focused on learning systems that adapt to several tasks, focusing on LML to progressively adapt to sequences of tasks while retaining and accumulating knowledge over time. We finally discussed the difference between offline optimization and online adaptation. In our work, we are interested in online adaptation for swarms of robots to unknown and possibly changing environments or tasks. As such, robots can potentially accumulate knowledge in an incremental manner, *i.e.* they may adapt over time to different experienced situations, while reusing previously learned behaviors in new tasks. Task and environmental changes may be cyclic, *i.e.* previous tasks or environments can reappear in the future. Consequently, instead of adapting to new tasks with no regard to old ones, it may be interesting for a robot swarm to remember how to solve past tasks, in case they reappear. We use Embodied Evolutionary Robotics approaches for swarms of robots to adapt online during actual task execution, in a never-ending learning scheme. In the next chapter, we provide an overview of Evolutionary Robotics approaches, that use Artificial Evolution to learn robot control, with a focus on Embodied Evolution to learn swarm robot behaviors.

3

Evolutionary Robotics

Contents

In this chapter, we present an overview of Evolutionary Robotics (ER), a general family of algorithms for the design of robot behaviors that take inspiration from natural evolution. We begin by reviewing the general approach of Artificial Evolution (AE) from the robotics perspective, while describing several components and their interplay in evolving robot control, as studied in the field of ER. We then review research on Evolutionary Swarm Robotics (ESR), with a focus on when, where, and how is evolution applied to learn swarm robot control. Finally, we present a classification and an overview of Embodied Evolution (EE) approaches to learn swarm robot control, in which the work in this thesis is inscribed.

3.1 Evolutionary Algorithms for Robot Control

Evolutionary Algorithms (EAs) are population-based stochastic metaheuristics that aim to find effective solutions for a given optimization problem. They progressively optimize a population of candidate solutions based on a measure of performance, or fitness value. This family of algorithms

relates to direct policy search (*cf.* Chapter 2) in that they do not require gradient information to progress, contrary to other ML algorithms such as backpropagation. Consequently, they are well-suited to problems where this gradient information is not available. EAs are loosely inspired by Darwinian natural evolution [Darwin, 1859], by adhering to its main principles: selection pressure and blind variations. EAs apply a set of evolutionary operators to iteratively search for better candidate solutions, or individuals. These operators apply selection pressure toward the best solutions in the population, and blind variations that explore the search space.

EAs are iterative algorithms that improve a population of candidate solutions until some termination criterion is attained. Each iteration of an EA is called a *generation*. The typical steps at each iteration are described in the following:

Initialization: the population is seeded with a set of initial candidate solutions. These solutions are typically random, although sometimes they may stem from *a priori* knowledge or a preliminary process of learning.

Evaluation: solutions are evaluated using a problem-dependent fitness function. Each solution is thus associated to a fitness value that measures its quality.

Selection: the fitness values are used to select a subset of the individuals in the population, so they can be copied to serve as parents for the next population. Selection pressure, as mentioned above, is applied here, and pushes the evolutionary search toward better solutions.

Variation: these selected parent solutions are modified using evolutionary operators to introduce new candidate solutions, the *offspring*. These operators perform the blind variations mentioned above, and are the source of novelty in the population of solutions.

Replacement: the newly evolved offspring solutions replace the current population, or part of it. Selection pressure is also applied at this step.

Termination condition: the algorithm iterates until a given condition is met. The algorithm returns one or a set of solutions, which may be part of the final population, or the best ever found.

Variations in the implementation of these steps five rise to many flavors of algorithms [Eiben and Smith, 2003]. ER algorithms follow this classical structure to optimize robot behaviors. We discuss the parts that play a major role in ER in the following.

3.1.1 Fitness Functions

Defining a fitness function to evaluate candidate solutions for a given task can be a challenging problem. The role of such a function is twofold. On the one hand, it is used to guide the evolutionary search, by defining a fitness landscape in the search space, to be navigated using the evolutionary operators. On the other hand, it provides a quality measure for the solutions resulting from evolution, *i.e.* it defines the goal to reach. In ER, fitness functions evaluate a policy by executing it on the environment and measuring the quality of the behavior resulting from the policy execution, *i.e.* they are direct policy search algorithms (*cf.* Chapter 2). However, fitness functions summarize information about how well the task is solved by the candidate solution in a single measure. Consequently, EAs must be able to exploit such limited information to guide the search process. Therefore, the results of an evolutionary process depend by a large extent on the definition of the fitness function.

In ER, fitness evaluation has several particularities. Typically, a given robot controller is evaluated by executing it in a given environment (real or simulated) for some time. During this time, the robot interacts with the environment, by receiving information using sensors, and producing actions on the environment using effectors. After this evaluation period, a fitness value is returned, which measures the performance of the behavior resulting from the robot-environment interactions. Fitness measures in ER are intrinsically noisy, due to several reasons: sensors and effectors are not perfect, initial conditions in which the robot is evaluated may vary, and the environment may be dynamic. Consequently, ER algorithms have strong requirements in terms of robustness: algorithms must cope with the noisy, delayed, poorly informative, and sometimes possibly misleading feedback provided by fitness functions. As such, designing appropriate fitness functions to provide adequate guidance to the evolutionary search has been recognized as a critical issue in ER for some time [Nelson *et al.*, 2009].

3.1.2 Selection Pressures

In natural evolution, selection pressure includes any factor that drives evolution toward individuals that are better fit to the environment. Consequently, in EAs, selection pressures correspond to any factor that influences the selection process for solutions to a problem, whether these factors are related to a goal-driven fitness function, or to auxiliary objectives aimed at improving the search [Doncieux and Mouret, 2014].

The strength of selection induced by a selection operator, with respect to the values of an objective function, is known as the intensity of selection pressure. The stronger the intensity of selection pressure, the less diverse the population of candidate solution becomes, and the faster it will converge. This can be measured by the takeover time, as introduced in [Goldberg and Deb, 1991]. The intensity of selection pressure plays a major role in the diversity of the solutions of an EA, which influences its results [Goldberg and Deb, 1991, Back, 1994].

In [Doncieux and Mouret, 2014], the authors review research in the ER literature, from the perspective of selection pressure. They classify selection pressures as either *goal refiners* or *process helpers*. The former are objectives that aim at defining or changing the optimum, or optima, of the fitness function. The latter are objectives that aim at improving the guidance of the search process so as to increase its efficiency, without changing the optima of the fitness function. In recent years, considerable effort has been made to investigate such auxiliary selection pressures, to cope with the intrinsic challenges of fitness functions in ER. For example, population diversity is known to reduce premature convergence to local optima in the fitness landscape [Squillero and Tonda, 2016]; several algorithms have been proposed to increase and maintain diversity in EAs: diversity search [De Jong *et al.*, 2001, Mouret and Doncieux, 2009b, Doncieux and Mouret, 2010], speciation and niching [Mahfoud, 1995, Goldberg and Richardson, 1987], crowding [De Jong, 1975, Manner *et al.*, 1992], or low-intensity selection operators [Mc Ginley *et al.*, 2011].

Another problem in ER that has been addressed using auxiliary selection pressures concerns the *reality gap* problem [Jakobi *et al.*, 1995]. This refers to transferring solutions evolved in simulation (*i.e.* where each candidate solution is evaluated by executing it using a simulator) to physical robots, which may introduce a difference in performance between the simulation and the real device. This is due to inaccuracies in simulators, which may be exploited by evolution, thus leading to unexpectedly poor behaviors on real robots. Nevertheless, the use of simulators is widespread, since it is cheaper and faster than evaluating every candidate solution on the actual robots. Consequently, a number of approaches have been proposed to reduce this reality gap, which exert selection pressure toward behaviors that perform well in real robots (either directly or indirectly) [Jakobi *et al.*, 1995, Koos *et al.*, 2012, Zagal and Ruiz-Del-Solar, 2007, Bongard and

Lipson, 2004].

A distinction can be made between two types of selection pressures in ER, concerning their nature related to given tasks:

- Environmental selection pressure, which implicitly drives the evolution of efficient behaviors based on environmental constraints.
- Task-driven selection pressure, which explicitly pushes evolution toward the given task, measured by a fitness function.

Environmental selection pressure can be thought of as a reproductive pressure toward individuals that are better fit to their environment, in the sense that they have higher chances of surviving in that environment. Here, surviving means to reproduce by spreading their genomes. This type of selection pressure is somehow closer to selection pressure in nature. In nature, fitness cannot be measured *a priori*, but is rather a result of a complex system of interactions throughout the lifetime of each living agent. Consequently, fitness can only be observed *a posteriori*, by measuring the proportion of individuals in the current population that derive from a common ancestor. Environmental selection pressure does not require a fitness function as it is typically seen in EAs, and is affected by many factors, including: morphological capabilities of the agents, *e.g.* related to movement; environmental harshness, *i.e.* how hard are the environmental conditions for agents to survive; or behavioral features, that may allow for agents that are adequately controlled to better reproduce.

On the other hand, task-driven selection pressure focuses on explicitly rewarding agents that perform well on a user-defined fitness function. This, in turn, provides a selective advantage in the EA, and fitter individuals are given a higher chance to survive to the next generation. Task-driven selection pressure has been the usual manner of inducing selection pressure in EAs since their origins, to optimize a user-defined objective function. In a way, task-driven selection pressure relies on *a priori* measures of fitness, which are measured before selection is performed. Additionally, both environmental and task-driven selection pressures can be combined to drive evolution toward agents that are able to survive in an environment while performing given tasks.

3.1.3 Neuroevolution

Selection pressures are applied to individuals in the population of the EA, in our case behaviors, which correspond to robot controllers. Consequently, an adequate controller representation much be chosen. Here, we discuss the notion of controller representation for EAs, with a focus on neuroevolution, *i.e.* the evolution of neural network controllers, which is the major approach in ER.

Controllers

The controller refers to the robot control system that, given sensor inputs and possibly a memory, computes the motor outputs that are executed by the robot. EAs distinguish between genotype, or genome, that encodes characteristics of a candidate solution to a problem; and phenotype, as the expression of the genome representing the solution itself, a robot controller in our case. On the one hand, variation operators act at the genotypical level, by modifying the genomes that represent candidate solutions. On the other hand, evaluation is performed at the phenotypical level, by measuring the performance of the solutions represented by genomes, in our case, robot behaviors resulting from the execution of the controllers. Consequently, choosing an adequate

representation for robot controllers, which allows evolution to efficiently find the best performing ones, is critical in ER.

Important properties of the controller genetic representation include, among others:

- Expressiveness, as the capacity of representing a wide set of controllers [Eberbach, 2002];
- Compactness, as the capacity to express complex controllers with relatively small genomes [Balakrishnan and Honavar, 1995];
- Modularity, as the capacity to represent separate substructures in the controller [Clune *et al.*, 2013];
- Evolvability, as the overall ability to reach many different genomes from any initialization [Konig and Schmeck, 2009];
- Smoothness of the induced search space, as the property of having representationally close genomes lead to functionally close controllers [Vassilev *et al.*, 2003].

The link between the encoding and the actual solution is called genotype-phenotype mapping [Fogel, 1995]. Such a map defines which phenotypical solution corresponds to a given genome. In EAs, adequately defining this mapping is essential, since it influences the evolvability of proper solutions, and approaches to evolve the mapping itself have been proposed [Chow, 2004]. As said before, variation operators, such as mutation and crossover, act on the genotypical space, while the evaluations used to apply selection pressure are done on the phenotypical space.

Many controller representations, as well as many corresponding genetic encodings and mappings, have been used in ER. Most works use neuroevolution, in which robot controllers are represented as Artificial Neural Networks (ANNs) [Floreano *et al.*, 2008]. This is also the controller representation used in our work. In the next section, we introduce the main aspects of neuroevolution in ER, including genetic encodings and corresponding variation operators, as well as the properties of this representational choice.

Neurocontrollers

Artificial Neural Networks (ANNs) are biologically-inspired computational models made of a set of interconnected basic units, the neurons. In these models, each neuron computes its activation as a simple function of the activation of the neurons connected to it, *i.e.* its presynaptic neurons, and the strength of the corresponding connections, *i.e.* the synaptic weights. The resulting overall computation of the ANN is distributed through its entire structure. Several types of ANN exist, and have been used in neuroevolution as robot controllers. These models have different computational properties and complexity. Types of ANNs include, for example, discrete McCullochs and Pitts neural models [McCulloch and Pitts, 1943], both feedforward, such as the multilayered perceptron, and recurrent, such as Elman neural networks, Continuous-Time Recurrent Neural Networks (CTRNNs) [Beer and Gallagher, 1992], reservoir computing networks, such as Echo-State Networks (ESNs) [Jaeger, 2008], spiking ANNs [Maass, 1997, Ghosh-Dastidar and Adeli, 2009], and Long Short-Term Memory (LSTM) models [Hochreiter and Schmidhuber, 1997].

All these ANNs variants possess several general intrinsic properties that are of use for robot controllers (*cf.* above). This is even more the case when such ANNs are evolved, *i.e.* built using EAs. Some of the desirable properties of evolving ANNs include:

- Theoretically proven universal approximation of functions with an arbitrary precision, provided some assumptions on the neural structure [Cybenko, 1989];

- Relative simplicity of encoding into a genetic representation;
- Generalization to similar but unseen conditions and robustness to noise in continuous analog inputs and outputs, *e.g.* robot sensors and actuators;
- Different possible levels of learning and adaptation: phylogenetic (evolution), developmental (maturation), or ontogenetic (lifetime learning);
- Smoothness of the search space: similar neural networks provide similar functional behaviors;
- Biological plausibility as models of control in natural systems, which allows for investigating biological theories with such artificial systems.

Additionally, many neuroevolution techniques have been developed to exploit the flexibility of ANNs, and thus improve the evolutionary search for performing solutions, either to accelerate the search, or to find better robot controllers. For comprehensive reviews of neuroevolution and related bioinspired techniques to develop neural networks, see [Floreano *et al.*, 2008, Kowaliw *et al.*, 2014, Risi and Togelius, 2017, Sher, 2012].

ANN Genetic Representation

Neural robot controllers consist of: a set of input neurons, corresponding to robot sensors; a set of output neurons, corresponding to robot effectors; possibly a set of internal or hidden neurons, that perform internal computations; and a set of weighted synaptic connections, which convey activation between neural units. Neuroevolutionary approaches can be divided into two main categories, depending on which part of the controller is evolved. On the one hand, a set of synaptic weights can be evolved. On the other hand, both the topology and the weights can be evolved, which is known as Topology-and-Weights Evolving Artificial Neural Networks (TWEANNs). Furthermore, depending on how neural networks are genetically encoded, there exist two main families of approaches. On the one hand, an ANN can be directly encoded as a directed graph that defines the structure and parameters of an ANN, *i.e.* direct encoding. On the other hand, the ANN may result from a process, interpreting a genome as a program that indirectly generates an ANN, *i.e.* indirect and generative encoding.

In direct encoding approaches, a graph representation of a neurocontroller is evolved. One possibility is to evolve the weights of a fixed structure, which is known as conventional neuroevolution. On the other hand, the neural structure can be evolved along with the corresponding weights. In all these cases, each part of the genome directly corresponds to a particular part of the ANN, and vice versa. Additionally, neural activation functions and other parameters of the ANN can be encoded in the genome.

On the other hand, indirect and generative encodings represent the genome as a program that specifies a process of development during the genotype-phenotype mapping. This program progressively creates an ANN by following the rules encoded in the genome. Such rules can be based on different paradigms, such as cell division, grammatical rules, or global neural connectivity patterns. There are several advantages of indirect encodings over direct ones. For example, they can be highly compact, thus allowing for increased scalability with respect to the size of ANNs; they can exploit modularity and neural module reuse, thus avoiding relearning of similar functions; and they allow to find and exploit useful regularities in the geometry of the neural functions, such as symmetries, periodicities, or repetitions with variation. Examples of approaches using indirect encodings in neuroevolution include Lindenmayer grammar systems [Kitano, 1990], cellular encoding [Gruau, 1992], Analog Gene Encoding (AGE) [Mattiussi and Floreano, 2007],

and Hyper-NEAT [Stanley *et al.*, 2009]. However, while the aforementioned properties clearly advocate for the use of indirect encodings, they raise different challenges: their computation may be expensive, thus limiting their application; parametrization of indirect encodings can be complex, thus requiring a costly preliminary parameter tuning procedure; and, while they efficiently cope with regularities in the task domain, it has been shown that slight irregularities can be challenging for evolution with indirect encodings [Clune *et al.*, 2011]. To alleviate these issues, approaches mixing indirect and direct encodings have been proposed, such as Hybridization of Indirect and Direct encodings (hybrID) [Clune *et al.*, 2009].

Variation Operators

Variation refers to the process of modifying solutions with evolutionary operators to explore the search space. These operators include mutation, which generates an offspring individual by slightly modifying the genetic material of a parent solution, and crossover, which mixes the genetic material of 2 or more parent solutions to potentially exploit their promising building blocks. Since variation is responsible for exploring the search space based on current solutions in the population, a good design of the variation operators is necessary for the algorithm to be efficient. Such operators should not be too disruptive (to preserve what was learned before), nor should they be too conservative (to avoid slow convergence and cycling around the same solutions in the search space). Additionally, the definition of both mutation and crossover operators depends on the corresponding genotypic encoding. In other words, unlike selection operators, variation operators are representation-specific, which means that there does not exist a set of universal mutation or crossover operators. Nevertheless, mutation operators for different encodings still share common properties, as do crossover operators.

Mutation operators aim at locally exploring the search space around the current selected parent solution, to find promising search directions, and progressively improve the solutions in the population. Mutation needs to be local, since excessively strong mutations are very disruptive, which results in the evolutionary process being close to a random walk over the search space. On the other hand, excessively conservative mutations slow down the progress in the search space and they may prevent evolution to escape local optima, which are common in complex optimization problems such as in learning robot control. Therefore, setting appropriate parameter values to regulate the strength of mutations is crucial in EAs. One approach consists in running a preliminary parameter tuning procedure to find such appropriate parameters for a given problem. There also exist techniques to adapt the strength of mutations on the fly, either using parameter-control algorithms (*e.g.* as step-size adaptation mechanisms in Evolution Strategies (ES) [Hansen and Ostermeier, 1996]), or by adding the parameters related to mutation to the genome encoding, and letting evolution find the right values over time, *i.e.* self-adaptive mutations [Kramer, 2010].

Crossover, on the other hand, aims at merging building blocks from a set of (typically 2) selected parents. Building blocks refer to subparts of the genomes that are linked to a phenotypic trait, *i.e.* parts that represent a functional feature of the solution corresponding to a genome. Thus, the goal of crossover is to integrate good properties from the parent genomes into one or more offspring individuals. In this sense, crossover operators are generally not local, since one crossover operation may result in a genome that is quite far from the parents in the search space, provided diverse enough parents. Crossover, if applied blindly, can be very disruptive: crossing two parents blindly may result in an offspring that does not retain the effective building blocks of the parents, but rather the ineffective ones. This is one of the reasons why crossover is not very widespread in neuroevolution: defining well-informed crossover operators for neural network encodings, able to choose and combine good properties of the parent neural networks, is challenging.

Variation operators for neurocontrollers are largely dependent on the encoding and the evolved features:

In fixed-topology evolving ANNs, typically, the genome is directly encoded as a vector of synaptic weights, and variation operators modify those weights. Mutation generally adds to the weights of the selected parent a value drawn from either a Gaussian random distribution parametrized by a step-size, σ, or from a uniform random distribution within a defined range. If the genetic encoding is binary, as in Genetic Algorithms (GAs), bit-flip mutations with some probability are also used. Crossover operators, if used, generally consist in either averaging the weights of the selected parents, or taking a subset of weights from each parent, to generate offspring.

In topology-and-weights evolving ANNs, in addition to similar weight mutation operators as in fixed-topology ANNs, structural mutations are added, such as adding or removing a neuron or a connection in the network with a given probability. Crossover operators are most of the time not used, since they may be disruptive and break structural building blocks. A remarkable counterexample is NeuroEvolution of Augmenting Topologies (NEAT) [Stanley and Miikkulainen, 2002], which provides a clever mechanism to allow for meaningful crossover. This is discussed in Chapter 7, where we present a set of experiment on neural topology evolution.

In indirectly-encoded evolving ANNs, there exist a large amount of different mutation operators, depending on the chosen encoding. For example, Hypercube-based NeuroEvolution of Augmenting Topologies (Hyper-NEAT) [Stanley *et al.*, 2009] abstracts the geometric connectivity pattern of an ANN using a higher-level network, called Compositional Pattern-Producing Network (CPPN). CPPNs are akin of ANNs, and are evolved using NEAT, with similar variation (mutation and crossover) operators (see topology-and-weights evolving ANNs above). Analog Gene Encoding (AGE) [Mattiussi and Floreano, 2007] defines network structures and parameters using an indirect, DNA-inspired sequential representation of symbols, which is decoded by interpreting those symbols in a developmental process similar to biological gene transcription. Mutations either modify, insert or delete symbols or fragments, or make a complete copy of the genome. Homogeneous crossover can be applied, which mixes fragments of two parent genome sequences, if those genomes are compatible enough. Different variation operators are used in other indirect ANN encodings.

A crucial question when using neuroevolution approaches to learn robot behavior is how to allow robots to progressively adapt their behavior, and develop skills over time. This question is central in our work, and, in the next section, we present different existing approaches in the literature of ER, we discuss their common features, and their differences.

3.1.4 Progressive Behavior Adaptation

When adapting to given tasks, robots may benefit from adequate prior knowledge, for example exploiting previous learning. One active research topic in ER concerns how to integrate successive learning stages in an efficient manner, *e.g.* how to provide priors from preliminary learning stages as a stepping-stone to learn to solve a final task. This has been addressed in the ER field with several related approaches, which carry different names: robot and fitness shaping [Dorigo and Colombetti, 1994], incremental evolution [Gomez and Miikkulainen, 1997], staged evolution [Téllez and Angulo Bahón, 2008], scaffolding [Auerbach and Bongard, 2011], never-ending evolution [Doncieux *et al.*, 2015], or open-ended evolution [Prieto *et al.*, 2010].

All these approaches share common features: learning of different skills is done sequentially, over time, and not simultaneously. One of the main focuses is set on providing an appropriate gradient for evolution to move from one stage to the following; and each evolutionary stage is considered more as a progressive process of adaptation, rather than a single-staged optimization process to build a system. On the other hand, the approaches mainly differ on the goals and techniques used to reach them. Some of them aim at building behaviors to solve a final complex task, by defining fitness functions that reward subgoals (shaping), by progressively modifying the environment or the robot morphology (scaffolding), or by defining an ordered syllabus of tasks to learn (incremental and staged evolution). Other approaches focus more on the long-term adaptation to given environmental conditions, by having a continuous online evolutionary process (never-ending evolution), while trying to develop increasingly complex and novel behaviors that adapt to possibly changing situations (open-ended evolution).

These approaches closely relate to a central question of our work: how to endow (swarms of) robots with autonomous adaptation capabilities using an evolutionary approach. That is, how to make robots capable of adapting to unknown, possibly changing environments, without requiring intensive explicit supervision. This question takes an additional dimension when considering swarms of robots, *i.e.* large groups of simple robots, that adapt together to given tasks in a shared environment. In the next section, we review work in the field of Evolutionary Swarm Robotics, that aims at automatically synthesizing swarm robot behavior with EAs.

3.2 Evolutionary Swarm Robotics

Swarm Robotics (SR) investigates how to design systems that include large numbers of simple robotic agents, so that they interact to perform a desired collective behavior [Bayindir and Şahin, 2007]. There exists a body of work in manually engineering swarm robot behavior [Brambilla *et al.*, 2013], which is typically based on trial-and-error approaches. However, these methodologies do not yet provide a generally applicable solution to the design of robot swarms. The main drawback of such engineering approaches is twofold: they may be extremely time-consuming; and they do not scale up efficiently with task complexity. Consequently, automatic optimization and learning approaches for the design of robot swarms constitute a promising alternative to alleviate these issues. Generally, researchers take inspiration from Nature to propose such automatic approaches to design swarm robot behavior, which is called bioinspired swarm systems. In this section, we present the main approaches in the literature to automatically build controllers for swarm robotic systems using EAs, *i.e.* Evolutionary Swarm Robotics (ESR).

Important algorithmic features regarding the automatic evolutionary synthesis of swarm robot behaviors (and robot behaviors in general) concern when, where, and how these behaviors are evolved [Bredèche *et al.*, 2010]. This refers to the temporal, spatial, and procedural perspectives of the evolutionary process:

- Offline (a preliminary phase of evolution) *vs.* online (evolution during actual operation).
- Offboard (evolutionary operators applied on an external computer) *vs.* onboard (evolutionary operators applied on the robots).
- Centralized EA structure (a single population) *vs.* distributed EA structure (separate subpopulations).

These features will be further discussed in Section 3.3, which concerns Embodied Evolutionary Robotics, the approaches used in our work to evolve swarm robot behaviors. Here, we start by

classifying the approaches with respect to the homogeneity of the controllers and behaviors displayed by the different robots (homogeneous *vs.* heterogeneous), and with respect to the level at which selection is applied (individual *vs.* team level), following [Waibel *et al.*, 2009]. We present existing work on evolving swarm behaviors, in both the homogeneous and the heterogeneous categories, with either individual or team selection, and discuss the lessons learned so far.

3.2.1 Homogeneous and Heterogeneous Control

ESR approaches with homogeneous control optimize robot controllers in a swarm where each robot carries a copy of the same controller. In these approaches, an EA maintains a population where each individual represents a controller that is cloned, and deployed on each robot of the swarm for evaluation. As such, they are often referred to as clonal approaches. Consequently, they mostly display homogeneous behaviors, or at least similar ones. Individual behaviors may be in some cases differentiated [Trianni and Nolfi, 2009a]. This may be due to situated specialization [Baldassarre *et al.*, 2003], either due to: perceptive differentiation, where each robot receives different perceptions, thus providing different control responses; internal dynamics, where the robots integrate information over time, thus diverging in their respective control responses; and lifetime online learning processes, where the controller of the robot is modified over time on the basis of environmental feedback.

Examples of work in ESR using homogeneous control approaches include: evolution of synchronization [Trianni and Nolfi, 2009b]; coordinated motion, collective decision-making, and self-assembly [Trianni *et al.*, 2014, Baldassarre *et al.*, 2004, Liu and Winfield, 2010]; evolution of acoustic communication [Ampatzis *et al.*, 2006, Tuci *et al.*, 2008], evolution of behaviors for modular robots [Hamann *et al.*, 2010, Hamann *et al.*, 2011]; evolution of path formation [Sperati *et al.*, 2010]; evolution of cooperation in homogeneous swarms [Baray, 1997]; evolution of self-organized specialization in foraging [Ferrante *et al.*, 2015]; task-allocation and task-switching [Tuci, 2014]; and evolution of self-organized communication networks with UAVs [Hauert *et al.*, 2009].

On the other hand, ESR approaches with heterogeneous control optimize robot controllers in a swarm where each robot carries a different controller. In these approaches, an EA maintains different genomes in the population, and these controllers are deployed simultaneously on the different robots in the swarm. Consequently, behaviors of individual robots in the swarm are heterogeneous by definition[2]. In some cases, cooperative or competitive coevolutionary approaches have been studied, where there are two or more separate coevolving populations [Nitschke *et al.*, 2012]. Controllers in these populations are deployed in different robots in the swarm. Such coevolutionary approaches may include one population per robot, or one population per subteam of robots. In other cases, especially in EER, where the EA is run onboard by each robot, the population is spread through the robot swarm, *i.e.* there is no particular *locus* where the entire population is stored.

Examples of work in ESR using heterogeneous control approaches include: evolution of modular self-assembling robots [Stradner *et al.*, 2009]; investigating the impact on cooperative foraging of heterogeneity and team or individual selection [Waibel *et al.*, 2009]; evolving swarm robot control using environmentally-driven decentralized Embodied Evolution [Bredèche *et al.*, 2012]; coevolutionary algorithms for robot swarms for phototaxis [Christensen and Dorigo, 2006] and RoboCup football tasks [Luke *et al.*, 1998]; evolving subpopulations for each team in foraging [Bongard, 2000]; investigating the conditions for the evolution of communication [Floreano *et al.*, 2007]; and decentralized Embodied Evolution for swarm robot online adaptation in a diverse set

[2]However, while being heterogeneous, they may be similar, especially if they share a common phylogenetic origin, *e.g.* if a parent controller generates several closely related offspring.

of tasks (phototaxis, navigation, self-assembly, aggregation, ...) [Watson *et al.*, 2002, Bredèche *et al.*, 2010, Silva *et al.*, 2012b, Haasdijk *et al.*, 2014, Bredèche *et al.*, 2015].

Evolving both homogeneous and heterogeneous control in robot swarms requires applying selection pressure to drive evolution toward better performing behaviors. This can be done at two main levels: using performance measures of each individual robot, or using performance measures of the entire swarm, of robots. We discuss these two mechanisms in the next section.

3.2.2 Individual and Team Selection

Along a different axis, works in ESR could be separated into two groups depending on the level at which selection is applied: at the level of individual robots, or at the collective level of a team or swarm. This refers to how the EA rewards behaviors in a swarm of robotic agents. On the one hand, individual-level selection rewards each robot controller on the basis of its individual fitness. On the other hand, team-level selection rewards each robot controller on the basis of the collective fitness obtained by the entire swarm (which may be homogeneous). This relates to the Credit-Assignment Problem (CAP) in multiagent systems: for individual-level selection to be applied, an adequate individual fitness function must be defined, which, in some cases, may be challenging: the individual contribution to a given collective task may be difficult to estimate. Conversely, team-level selection can be applied by using an aggregate or accumulated fitness measure over the entire team, thus delegating the CAP to the EA. In this case, since the contribution of each robot to global performance is not available, the EA must cope with this coarse-grained information to evolve performing behaviors.

In [Waibel *et al.*, 2009], it was shown that, when evolving cooperation in a foraging task, homogeneous controllers using team selection provided the best results. However, both controller homogeneity and team selection pose strong constraints for evolution to be applied in a realistic situation: it requires a central system that is able to communicate with all the robots in the swarm. Furthermore, homogeneous control ESR make the evolution of specialization challenging: for subgroups in the swarm to show specialized behaviors, there must be an algorithmic mechanism that allows for differentiation, which is not the case, at least regarding evolution[3], in ESR with homogeneous control.

3.2.3 Generalization, Specialization and Division of Labor

Questions on the evolution of behavior patterns that are particularly collective, such as cooperation, specialization, or division of labor, have received considerable attention in ESR [Jones and Matarić, 2003, Ferrauto *et al.*, 2013, Bernard *et al.*, 2016]. Sometimes, these studies aim at answering a question about the biological mechanisms in nature through which such types of behavior may arise, by using evolving robot swarms as a model for collective systems in nature. In other cases, the goal of these studies concerns how to automatically design swarm robotic systems to solve given tasks that may require, or benefit from, such collective behaviors, *e.g.* specialization. These two goals are not opposed, and can benefit from bidirectional synergies: a better understanding of natural collective systems can lead to a better design for artificial ones, and the design and experimentation on robotic collective systems may help proposing hypotheses for natural ones.

For example, in a given environment, robots may need to learn different tasks with an ESR approach. Roughly, this can be done in two ways: either all the robots evolve generalist behaviors

[3] As said before, situated specialization could arise in homogeneous control evolution, *e.g.* through lifetime learning mechanisms or perceptive differentiation.

that perform all tasks (generalization); or the swarm evolves subgroups of specialist robots that can perform one task each (specialization). Specialization refers to the situation where one or more robots in the swarm becomes an expert in a particular activity or task among the set of tasks to be addressed [Ferrauto *et al.*, 2013]. The conditions for the emergence of specialist robots has been an important topic of research in the field of Swarm Robotics from its beginning [Jones and Matarić, 2003]. An important concept related to swarm robot specialization is division of labor [Labella *et al.*, 2006], *i.e.* how individual robots self-organize in a decentralized manner to fairly allocate tasks to robots [Gerkey and Matarić, 2004, McLurkin and Yamins, 2005, Ducatelle *et al.*, 2009]. Examples of research on specialization and division of labor in swarms include: mechanisms for the regulation of division of labor [Bonabeau *et al.*, 1996]; specialization through reinforcement learning mechanisms in homogeneous multiagent systems [Murciano *et al.*, 1997]; quantitative assessments of diversity specialization in learning swarms [Li *et al.*, 2004]; impact analysis of genetic relatedness and agent interactions on the division of labor in swarms [Waibel *et al.*, 2006]; emergence of collective specialization with neuroevolution [Eiben *et al.*, 2006, Nitschke *et al.*, 2012]; impact of social influence on specialization [Cockburn and Kobti, 2009]; self-organized specialization and generalization for energy foraging in robot swarms [Kernbach *et al.*, 2012]; an impact analysis of different EAs on the evolution of specialization or role allocation [Ferrauto *et al.*, 2013]; the conditions for behavioral specialization in robot swarms using distributed Embodied Evolution [Trueba *et al.*, 2013, Montanier *et al.*, 2016, Bredèche *et al.*, 2017]; or the emergence of task specialization and partitioning using Grammatical Evolution with group selection [Ferrante *et al.*, 2015].

In this thesis, we investigate a particular family of evolutionary approaches to automatically design swarm robot behavior, Embodied Evolutionary Robotics, which considers a group of robotic agents, each running an EA onboard to evolve behaviors *in situ* while operating. In the next section, we describe the main features of this family of approaches, while providing a classification related to how inter-robot interactions influence the EA. We then discuss existing work in Embodied Evolutionary Robotics, and the lessons learned so far.

3.3 Embodied Evolutionary Robotics

Embodied Evolutionary Robotics (EER), or Embodied Evolution (EE), is concerned with building control systems for robots that adapt online and progressively to given environments. The EE methodology is closely related to evolution in nature, in which organisms reproduce and evolve in a distributed and asynchronous manner, without any external authority orchestrating their development. The origin of these approaches lies in a seminal work by Watson *et al.* in the early 2000s [Watson *et al.*, 2002]. In that paper, the authors describe a small swarm of physical robots that learn to perform phototaxis (approaching a light source), where each robot locally runs an EA and they exchange genetic material. Following this work, several authors continued research using similar approaches to evolve swarm behaviors for different tasks, as well as for testing biological hypotheses, such as the conditions for the evolution of altruism [Montanier and Bredèche, 2011b].

In EE, each robot runs a separate EA onboard, with a local population, and evolutionary operators, such as evaluation, selection and variation, are run by each robot. Thus, this can be seen as distributing an EA over a swarm of robots. EE takes place online, while robots are deployed for the actual task. This is opposed to offline approaches, typical of traditional centralized ER, in which there are two separate phases: a preliminary learning phase, during which evolution optimizes behaviors in the population; and a deployment phase to exploit the

learned behaviors, during which the behaviors remain fixed. Consequently, EE approaches are open-ended, *i.e* evolution never stops, in a never-ending process of adaptation. As such, these are potential candidates for adaptation to changing environments and tasks. Further, reality gap issues, as well as transfer issues related to the (possibly physical) evolution, in controlled environments, can be sidestepped, since EE approaches advocate for an evolutionary process running on the robots during operational time. Additionally, controller evaluation in EE for robot swarms is intrinsically parallel and asynchronous, thus accelerating the evolutionary process with respect to serial fitness evaluations, as is often done in traditional offline ER. An overview of EE can be found in [Bredèche *et al.*, 2010] and [Bredèche *et al.*, 2015].

Although EE possesses the aforementioned intrinsic advantages, it also raises several challenges that must be considered, which are listed here.

Local selection, variation and evaluation. Since each robot runs a separate EA, individuals in the respective local populations are different. As such, selection and variation act on partial and local views of the global population of the swarm. Consequently, this changes the dynamics of evolution, and may lead to a slower convergence as compared to centralized ER algorithms. Additionally, each robot is responsible to evaluate its own controller. There is no global observer that could provide an external fitness measure for a given behavior. Consequently, attention must be put into designing informative fitness functions that can indeed be locally computed by the robots themselves.

Variable starting conditions and noisy evaluations. Since controllers are evaluated starting from the position where the previous controller ended its evaluation, fitness estimates may be unfair. For example, in a navigation task, if a robot ends its evaluation in a tight space, the next controller to be evaluated must escape that space before being able to improve its fitness. To alleviate this issue, some authors define a period at the beginning of each evaluation, during which the fitness is not measured, to allow robots to recover from disadvantageous situations. This is known as maturation time [Wischmann *et al.*, 2007] or recovery period [Elfwing *et al.*, 2011]. Additionally, since each robot evaluates its fitness from different starting conditions, and robots share the same environment, the evaluation conditions are highly variable. Consequently, fitness estimates by the robots can be extremely noisy. Reevaluation of past controllers is a widely used and effective technique to help in reducing noise.

Unary replacement. Since at any moment each robot can only run and evaluate a single controller, replacement operators must carefully choose which one to evaluate. Consequently, EE uses time-sharing mechanisms on each robot to evaluate different controllers over time: a robot divides its execution time among the genomes that it carries. This may slow down evolution, since, generally, the population in a new generation differs in a single individual from the previous generation.

Parameter adaptation. Setting the appropriate values for the parameters of an EA (or adapting them during the run) has been shown to be critical to the performance of the evolutionary search. However, EE approaches operate in different conditions, and the same methods used in centralized ER settings may not be as performing in EE. This requires special parameter adaptation techniques, that take into account the decentralized nature of EE.

Distributed communication and computational limitations. In some EE approaches, including the ones in this thesis, robots exchange genetic material when meeting. These approaches fall in the family of distributed EE, and they ensure genome spread by communicating with each other, which has has several benefits (*cf.* Section 3.3.2). However, relying on communication adds further constraints and needs to the systems: a local communication system with robust communication protocols needs to be designed. Further, since EE approaches are meant to be deployed in swarms of simple robotic agents, they should not be based on complex computations, and must rely on simple mechanisms with low computational cost, in terms of time and resources.

Possible robot endangerment. Since evolution takes place during operational time, *i.e.* once the robots are deployed, there is a possibility for the search to evaluate candidate solutions that are physically dangerous for the robot. These behaviors should be identified, either *a priori* or as soon as possible, and the corresponding individuals should be removed from the populations.

These challenges are considered in existing work in EER. For example, simple mechanisms are typically implemented, given the computational limitations of individual robots in swarms. EER approaches have been developed over almost two decades, and have received different names, such as asynchronous evolution, embedded evolution, or online onboard distributed evolution. The current common agreement on the term Embodied Evolutionary Robotics stems from the fact that the evolutionary operators are *embodied* on each robot, *i.e.* they do not rely on an extrinsic central authority orchestrating the evolution of behaviors over the robot swarm. In the rest of this chapter, we provide a taxonomy of work in EER, following [Eiben *et al.*, 2010a]. Works in EE are divided into encapsulated, distributed, and hybrid approaches, depending on how the local populations of each robot are managed. We classify a list of research in EE into those three categories, while discussing the main findings in the field.

3.3.1 Encapsulated EER

In encapsulated approaches to EER, each robot runs an entirely autonomous EA onboard, which includes an internal population of candidate controllers. Robots do not communicate with each other, and the evolutionary process is akin to a set of parallel instances of an EA. Nevertheless, robots share the same environment, so, even if there is no explicit communication, they may physically interact, *e.g.* by colliding or by sensing each other. At any moment, each robot runs an active genome, while maintaining an entire local population, which is isolated from local populations in other robots. These approaches do not explicitly require a swarm of robots, being applicable in single robot schemes, where the EA is run onboard. Examples of work on encapsulated EER include the following.

Balance between initial training and online adaptation

In [Walker *et al.*, 2006], the authors study the interplay of initial training as a prior for further lifelong embodied adaptation. In a set of experiments with a single robot, they investigate the impact of having an initial learning phase and a lifelong online adaptive phase on the performance of an evolving robot. They conclude that both phases have a significant impact when evolving performing behaviors that are able to adapt to dynamic environments.

Coping with noisy evaluations and changing conditions

In [Bredèche *et al.*, 2010], the authors propose $(\mu+1)$-ONLINE EA, in which each robot maintains a population of μ individuals, and several mechanisms to address intrinsic issues of EE are proposed and tested. Adaptive step-size Gaussian mutation is applied to the weights of an ANN, to avoid premature convergence (*cf.* parameter adaptation above); a recovery period at the beginning of each evaluation is included to allow the robots to escape possibly dangerous situations; and the best genome in each local population is probabilistically reevaluated in order to cope with noisy evaluations (*cf.* variable starting conditions and noisy evaluations above). Further, in [Eiben *et al.*, 2010b], the authors use self-adaptive mutation operators to adjust the mutation step-size parameter of the EA during evolution, as in Evolution Strategies. The step-size is encoded in the genome, and is subject to evolution. The authors conclude that the choice of the self-adaptation mechanism is important, since such mechanisms do not all perform in a similar manner. In [Montanier and Bredèche, 2011a], the authors investigate how to further avoid premature evolutionary convergence. The authors propose the $(1+1)$-Restart-Online Adaptation Algorithm, which builds on the previous algorithm by adding a restart mechanism. The restart mechanism randomly reinitializes the internal population whenever the evolutionary search has converged. The goal of such a restart mechanism is to help in avoiding getting stuck in local optima, thus performing a more global search. The authors conclude that the algorithm efficiently addresses a trade-off between local and global search, thus providing wider exploration of the search space.

3.3.2 Distributed EER

In distributed approaches to EER, each robot carries a single genome as its local population, corresponding to its current active controller. Robots broadcast their respective genomes and fitness values to nearby robots, which allows them to fill internal temporal lists. Selection is locally performed based on this local list of genomes gathered during the previous generation, after which, the temporal list is emptied. Consequently, in these approaches, the global population of the EA (to which each individual robot has no access) corresponds to the robot swarm, or more specifically, to the set of active genomes of each robot in the swarm. As such, evolution emerges from local interactions between evolving robots, which may be seen as helping each other learn. This could be related to social learning approaches in Multiagent Systems (MASs) [Montes de Oca and Stützle, 2008]. Examples of work on distributed EER include the following.

A first distributed EE algorithm

In this seminal paper [Watson *et al.*, 2002], the authors propose and experimentally validate a first distributed EE algorithm, Probabilistic Gene Transfer Algorithm (PGTA). This algorithm is distributed over a set of physical robots, that broadcast genes (parts of their respective genomes) at a rate proportional to their respective fitness. Upon reception, a robot integrates received genes into its genome with a probability inversely proportional to its own fitness: the less performing the current robot behavior, the higher the probability to integrate received genes. The authors, who coined the term of *Embodied Evolution*, perform a set of experiments on a phototaxis task with a swarm of 8 physical robots, showing the feasibility of the distributed EE approach. Further, they discuss some of the intrinsic advantages of EE, such as completely avoiding the reality gap, as well as accelerating evolution through parallel evaluations, and they presented the approach as opening new ways of research for collective robotics and physical Artificial Life (Alife).

Maturation time to develop behavioral skills

In [Wischmann *et al.*, 2007], the authors investigate the influence of maturation time on robot performance (*cf.* variable starting conditions above). Maturation refers to a transient period at the beginning of each evaluation, during which the behavior of each robot has no influence on its fitness. The underlying rationale is that, by providing some time for each controller to develop its behavioral skills without being penalized, a more accurate fitness estimate can be measured. Given that each controller evaluation on a robot starts from the situation in which the previous evaluation finished, such a maturation time can allow effective controllers to move a robot away from a possibly dangerous or challenging situation resulting from the behavior of its predecessor. The authors conclude that choosing an adequate duration for the maturation period (neither too long, nor too short), largely influences the performances reached by evolution.

Environment-driven evolution of survival

In [Bredèche *et al.*, 2012], the authors investigate the evolution of behaviors that allow for survival on the physical vehicles, *i.e.* the robots, in the absence of any explicit fitness function. They propose an EE algorithm, called minimal Environment-driven Distributed Evolutionary Adaptation (mEDEA), a simple distributed EE algorithm that, instead of relying on an explicit fitness measure, relies on the environmental conditions to shape the evolved behavior, in an Alife-like system. The authors show in a series of simulation experiments that, even without explicit selection pressure, robot swarms can evolve behaviors for genome survival in a distributed manner. In one of the experiments, robots evolved navigation capabilities: the genomes of robots that moved faster had more chances of spreading their respective genomes to other robots, thus having more chances of being selected afterwards. In another experiment, a "sun" was placed in the environment, and the robots perceived its orientation and distance. The position of the sun was changed during the experiments. The robots evolved behaviors in which they reach different types of consensus, as a way to meet other robots and reproduce: mainly they either approach the sun, or flee from it. They further performed a set of successful experiments with physical robots, which also raised a number of technical issues to be considered when applying EE in actual robots. In [Montanier and Bredèche, 2011b, Montanier and Bredèche, 2011c, Montanier and Bredèche, 2013], the authors investigate how altruistic behaviors can emerge in a swarm of robots that evolve using such an environment-driven distributed EE, *i.e.* mEDEA. Altruism is a special type of cooperation, in which there exist individual robot behaviors that sacrifice their own survival to benefit other robots. The authors study the emergence of such altruistic behaviors in different settings, *e.g.* the evolution of altruism when facing the tragedy of the commons, a situation where altruism is necessary for the entire population to survive; or how spatial dispersion influences the evolution of altruism.

Combination of environmental and task-driven selection

In [Haasdijk *et al.*, 2013], the authors investigate the impact on survival and task skills of both environment-driven selection, as in mEDEA, and task-driven selection, guided by a fitness function. The authors propose an algorithm, called Multi-Objective aNd open-Ended Evolution (MONEE), that combine both features and allows for efficient evolution of behaviors for concurrent foraging tasks, *i.e.* with different types of food items. The algorithm also adds a "market" mechanism that regulates division of labor over time, by changing the fitness value of different subtasks, or types of item to be foraged, as a function of their respective availability: the rarer the type of item, the higher its value. The authors conclude that this mechanism allows the swarm to

avoid neglecting types of items that could be more difficult to collect, over easier, more common types. Other improved market mechanisms to regulate division of labor are proposed in further works, such as in [Haasdijk and Bredèche, 2013], where users can define a bonus or malus for each task, in order to influence their value; or in [Bangel and Haasdijk, 2017], where a market scheme based on a sigmoid function is proposed to further amplify selection pressure toward rarer tasks. There has been a body of work based on MONEE that investigates how to efficiently combine environment-driven and task-driven selection pressures. For example, in [Haasdijk *et al.*, 2014], the authors perform a thorough analysis of the interplay of both types of selection pressures in a foraging task with two types of items. They focus on the degree of environmental adaptation, *i.e.* the ability to survive in the environment, and task efficiency. The authors analyze the impact of the market mechanism on effort distribution in the swarm when addressing two simultaneous tasks that are unequally available. They conclude that, while the algorithm allows for a more equal task share than without the market mechanism, the proportion of individuals in the swarm that specialize in the more difficult task remains lower than the proportion of corresponding available food items.

Environmental influence in distributed EE

In [Steyven *et al.*, 2016], the authors investigate how some environmental parameters influence the evolved behaviors using a variant of mEDEA. They perform a set of experiments in a foraging task, while varying the value and number of food sources in the environment, and analyze the resulting fitness landscapes. The authors provide a detailed discussion on the different regions of such landscapes, and conclude that more effort should be put on the definition of environmental parameters in EE experiments. They further claim that neutral regions, *i.e.* regions that provide a balance between resource availability and consumption, could be of highest interest to perform experiments, since it provides enough selection pressure without being extremely harsh for robots to survive. In [Steyven *et al.*, 2017], the authors investigate the interplay between distributed EER and individual lifetime learning mechanisms in a set of dynamic foraging environments. They conclude that different environmental conditions, dynamical changes, and combinations of learning and evolution can lead to drastically different evolved behaviors. Particularly, they show that evolution with individual learning allows for adaptation to changing environments, especially when the learning opportunities are more frequent.

Influence of social learning in EE

In [Heinerman *et al.*, 2015a], the authors investigate a robot swarm with three levels of adaptivity: (embodied) evolution, individual learning, and social learning. The authors propose an algorithm combining a distributed EE algorithm for the sensory layout; an internal lifetime learning algorithm ($(1+1)$-ONLINE ES) to learn the neural controller parameters during the lifetime of a robot; and a social learning mechanism that allows a robot to communicate its *memotype* (which includes the currently learned ANN parameters) to other robots to enhance their learning. The authors report a series of experiments in simulation with different settings, and conclude that social learning increases both the quality and the speed of learning. In [Heinerman *et al.*, 2015b], the authors investigate similar questions in a series of experiments using a set of 6 Thymio II physical robots augmented with an additional processor (Raspberry Pi), a WiFi dongle, and an additional battery. Their conclusions are similar to their previous work: social learning mechanisms improve the speed of learning and the quality of the solutions. Additionally, the authors remark that there is some degree of agreement on the evolved sensory layout. They claim that

this is due to social learning having an impact on increasing selection pressure to evolve the sensory layout.

Embodied Evolution of specialization

In [Trueba *et al.*, 2013, Montanier *et al.*, 2016], the authors investigate which environmental and algorithmic features favor the evolution of specialized behaviors in robot swarms when using distributed EE. In those works, the authors experimentally test different conditions with respect to a set of tasks that are simultaneously available in the environment. They evaluate the impact of such conditions on the level of specialization in different robots. In [Trueba *et al.*, 2013], the authors propose a generic distributed EE algorithm, and thoroughly analyze, both theoretically and on real robots, the influence of a set of canonical parameters in the algorithm with respect to evolved specialization in a robot swarm. They conclude that the replacement probability and the ratio between exploration and exploitation are the most influential paramenters regarding specialization, and they provide some guidelines to adjust them. In [Montanier *et al.*, 2016], the authors investigate the influence of environmental conditions on the evolution of specialization in distributed EE when generalist behaviors are not possible. They conclude that reproductive isolation and a large population size are essential for specialization to evolve. Reproductive isolation includes spatial and temporal separation, but also mating choice. In a recent work in [Bredèche *et al.*, 2017], the authors study the impact of selection operators on the evolution of specialization. They conclude that proportionate selection promotes the evolution of specialization in distributed EE. This is even more the case when there is reproductive isolation and large population sizes, as in the work in [Montanier *et al.*, 2016].

3.3.3 Hybrid EER

Hybrid approaches to EER combine features from the two previous categories. Each robot runs an entire EA, which includes an internal population of controllers. Additionally, robots communicate their respective current controllers and fitness measures when meeting, which allows receiving robots to enrich their local populations. Consequently, there is a set of separate evolutionary processes, that communicate with each other, in a scheme that is akin to migration in spatially-structured EAs. A non-exhaustive list of examples of work on hybrid EER includes:

Evolution of learning skills in groups of robots

In [Elfwing, 2007], the authors investigate the evolution of learning mechanisms and parameters, inside a population of robots. In their work, Reinforcement Learning algorithms are coupled with EE to develop behaviors in a swarm of robots. Robots exchange genetic material when meeting, which may include both learning and controller parameters, while keeping internal local populations for each robot, and use time-sharing mechanisms to evaluate the genomes in each robot's subpopulation. Part of the experiments are performed in simulation, and part in real robots, while sometimes initiating evolution in simulation, and deploying it into real robots for the last generations. In this work, the authors focus on the interplay of evolution and learning processes, and thoroughly analyze the evolved learning parameters. One of the main conclusions is that evolving learning parameters can provide a drastic improvement to learning, in terms of speed, quality of behaviors, and need for exploration in the learning algorithms.

Self-adaptation of fitness evaluation time

In [Dinu *et al.*, 2013], the authors propose and evaluate a self-adaptation mechanism for the evaluation time using a hybrid EE algorithm. The evaluation time corresponds to the duration of the evaluation phase of each controller, and is known to have a strong impact in EE. In this work, each robot keeps an internal population, while a newscast communication algorithm is used for inter-robot genome migration. The self-adaptation mechanism relies on encoding the evaluation time into the genomes, using an exponential encoding. The authors perform a set of experiments addressing foraging in both static and dynamically changing environments, and conclude that self-adapting the evaluation times provides effective and adaptive evolution, while avoiding the burden of a preliminary parameter-tuning phase.

Neural topology evolution in EE

In [Silva *et al.*, 2012b], the authors present online distributed NeuroEvolution of Augmenting Topologies (odNEAT), a hybrid EE algorithm that evolves the topologies and weights of neural controllers. odNEAT adapts the neuroevolutionary NEAT algorithm to the online decentralized settings of distributed EE. Each robot maintains an internal population, which is updated with received genomes if they have a high enough fitness. A local fitness-sharing (niching) scheme is applied, and a tabu list is kept to avoid recently evaluated poor genomes. The niching mechanism relies on computing a pair-wise distance between the structure of the ANNs in the local population. The algorithm uses high-precision timestamps to mark the evolving neurons and connections, to allow for such structural distance computations. They perform a set of experiments on an aggregation task where robots need to move close to one another. The experiments show the effectiveness of odNEAT in evolving high performing behaviors. Further, the authors run an additional set of ablation studies to evaluate the impact of each algorithmic component, and concluded that all of them are necessary for effective distributed evolution of topologies and weights in neurocontrollers for swarm robot behavior. In further works, the authors use odNEAT to evolve neuromodulated controllers [Silva *et al.*, 2012a]; accelerate evolution with odNEAT using modules, called macro-neurons [Silva *et al.*, 2014]; and combine odNEAT with a hyper-heuristic, Online Hyper-Evolution (OHE), that allows to adapt the search on the run by choosing the best algorithm to use at every moment [Silva *et al.*, 2016]. We use odNEAT in our experiments on topology evolution in Chapter 7, where we describe it in detail.

3.4 Conclusion

In this chapter, we have provided an overview of the algorithmic framework of Evolutionary Robotics, with a focus on Evolutionary Swarm Robotics for large groups of robots. We have further discussed Embodied Evolutionary Robotics, which are the methods used in our work to evolve swarm robot behavior. Concretely, in our work we use distributed and hybrid EER algorithms, both considering local interactions and communication between robots with respect to the evolutionary process. In the remainder of this thesis, we present our contributions to the distributed embodied evolutionary adaptation of behaviors in swarms of robotic agents.

Part II

Distributed Adaptation of Swarm Robot Behavior

4

Approach for Swarm Robot Adaptation

Contents

In this chapter, we introduce and discuss our general approach to adaptation of swarm robot behavior using distributed Embodied Evolution (EE) algorithms, presented in Chapter 3. Such distributed algorithms are run by each robotic agent in the swarm, and they rely on this parallel execution to progress. Controller evaluations on different robots are performed simultaneously, which accelerates and improves evolution, while providing the algorithm with a diversity of experienced situations. Although potential benefits of distributed EE approaches are numerous, some of them remain to be tested. Concretely, as seen in the previous chapter, since an EE algorithm is run online, there is no separation between learning and deployment phases, which is the case in offline algorithms. Consequently, EE algorithms potentially allow for controller adaptation to changing conditions. When a swarm of robots needs to adapt to different conditions online, it is of paramount importance for this adaptation to be efficient, *i.e.* the swarm needs to rapidly achieve high performing behaviors when changes occur. When a new situation is presented to the swarm, adaptation may erase previous knowledge (*cf.* catastrophic forgetting, Section 2.1.1). However, the situations to which a swarm adapts are likely to repeat. Consequently, retaining previously learned tasks can increase the efficiency of readaptation, since learning from scratch is avoided.

In this chapter, we present the common properties of the experiments in our work. This includes the baseline EER algorithm, which is instanciated on each set of experiments, the morphology setup for each robot in the swarm, including the sensor and motor layout, and a description of the environments used in our work. We introduce a set of proposed metrics that integrate information over time taking into account the online nature of EER. These measures are used in the experiments of the following chapters. We discuss the use of simulations in our experiments

on swarm behavior evolution, and we present and describe Roborobo!, the simulator that we use. Subsequently, we present our targeted scenario for sequential adaptation to unknown and possibly changing conditions. We describe the problem of forgetting previously acquired knowledge in such a sequential learning, while distinguishing between two levels of forgetting: individual and populational. Finally, we present an overview of the research questions of this thesis, and our contributions that answer them, while describing how they align regarding adaptation to unknown and possibly changing environment and tasks for swarms of robots.

4.1 Baseline Algorithm: vanilla EE

Algorithm 4.1 shows the baseline algorithm that we consider in our work. It is a simple distributed EE algorithm, based on mEDEA, in which each robot maintains an active genome, randomly initialized at the beginning of evolution. The active genome is executed and evaluated for T_e timesteps, *i.e.* an evaluation period, by mapping it to the corresponding neurocontroller. At each timestep, the ANN controller is run by computing the current outputs from current sensory inputs. The outcome of the robot behavior is used to update the fitness value of the active genome. During the evaluation period, robots locally broadcast their respective active genomes and fitness values to other nearby robots, which store them in a local list or population. If the same genome is received several times during a generation, its fitness value is updated; *i.e.* multiple copies of the same genome are not allowed in local lists.

Once T_e timesteps are elapsed, the generation is finished. The active genome and its corresponding fitness value are added to the local population, which allows for isolated robots that do not meet other robots to continue evolving. At this moment, the robot needs to choose a new active genome for the following generation. To do so, it selects a genome from its local list, possibly based on its fitness value. Subsequently, the selected genome is mutated (typically using a Gaussian mutation on all the synaptic weights of the neurocontroller), and it replaces the active genome. The local list of each robot is emptied when starting a new evaluation period, so selection is done on the genomes that a robot collected during the previous generation. As such, which genomes are in the local list of each robot depends on the encounters of the robots in the swarm, as defined by their behaviors. At this moment, the evaluation of the new controller begins.

Algorithm 4.1: Vanilla distributed Embodied Evolutionary algorithm. It corresponds to mEDEA [Bredèche *et al.*, 2012] with task-driven selection pressure, and includes self-insemination and generational emptying of the local list.

```
1  g_a ← random()
2  while true do
3      l ← ∅, f ← 0
4      for t ← 1 to T_e do
5          exec(g_a)
6          f ← evaluate()
7          broadcast(g_a, f)
8          l ← l ⋃ listen()
9      l ← l ⋃ {(g_a, f)}
10     selected ← select(l)
11     g_a ← mutate(selected)
```

Robot Morphology

In all our simulated experiments on swarm robot behavior evolution, we use a similar robot architecture, which is depicted in Figure 4.1. Robots are circular, and perceive the environment using sets of 8 sensors evenly-spaced around the robot. These sensors may provide different informations, depending on the experiment, *e.g.* proximity to obstacles, proximity to food items, or proximity to other robots. The exact set of sensors for each experiment is presented in the corresponding experimental settings (see Chapters 5, 6, 7). Robots move using two differential-drive wheels, left and right of the robot, which may have positive or negative activation to move forward of backward. The lamp in the center of the robot, used only on the experiments of Chapter 6, is an additional actuator that displays a color as determined by the robot controller.

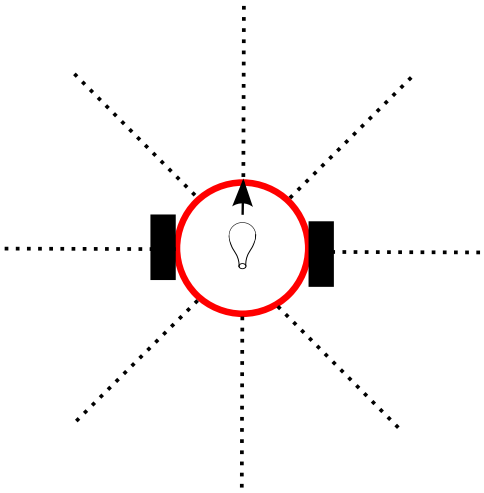

Figure 4.1 – Top view of the morphology for the robots in the swarms of the experiments in this thesis. It includes the sensory layout (orientation and range, dotted lines), the differential-drive wheels, which move the robot (black rectangles), and the circular shape of the robot. The lamp in the center of the robot is only used in the experiments of collaborative item collection (see Chapter 6).

Environment

In all the experiments on swarm robot evolution of this thesis, a swarm of robots is deployed in a bounded environment (see Figure 4.2), possibly containing obstacles (black lines in the figure). Robots address tasks of two main categories: navigation with obstacle avoidance or item collection. In the navigation with obstacle avoidance task, the robots need to move as fast and straight as possible, while avoiding static and moving obstacles. In the item collection task, the robots need to approach and collect food items that are spread in the environments (green circles in the figure).

Online Performance Metrics

A characteristic of online ER is that robots continuously adapt while performing the task. Consequently, the performance obtained over time during such adaptation is actually the performance on the real task. It follows that the best fitness reached during evolution is not a reliable measure of the global performance of the algorithm, since it only reflects a level of performance at

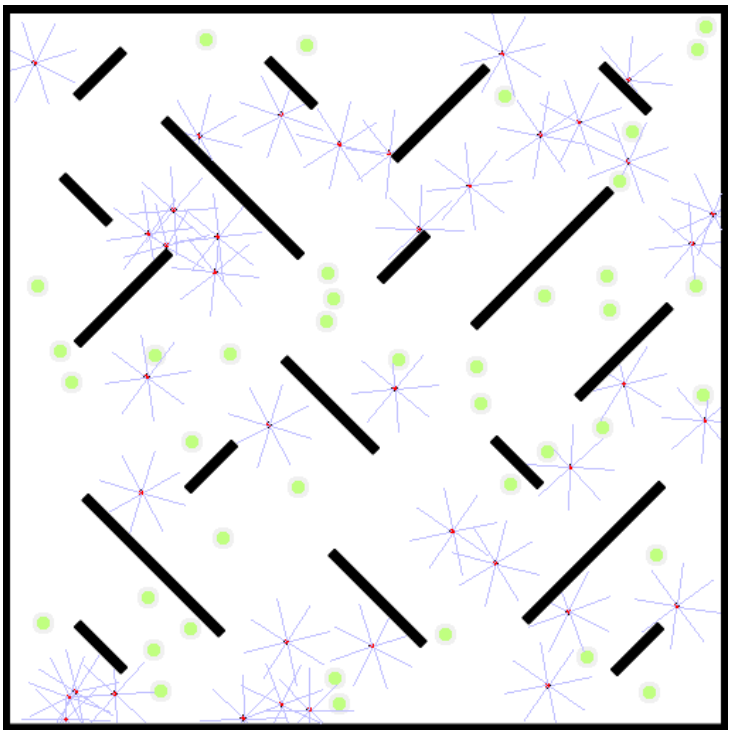

Figure 4.2 – An example of simulation environment used in the experiments of this thesis. It contains robots (red dots with thin hairlines representing sensors), obstacles (dark lines) and food items (green dots).

one point of the evolution. Furthermore, fitness evaluation is inherently noisy, due to different evaluation conditions encountered by the robots. Consequently, based on the sum of individual fitness of the robots in the swarm (*Swarm Fitnes*), we propose four metrics for online evolution that summarize information on the swarm spanning over several generations. They are used only for post-evaluation and comparison in our experiments, and are computed once the evolution has ended. A pictorial description of these four measures is shown in Figure 4.3. Two parameters are defined in each set of experiments to compute the measures: a computational budget in terms of a percentage of the duration of evolution (*budget*); and a target swarm fitness level (*target*), as a percentage of the maximum level of performance ever reached over all the runs of a set of experiments. Our proposed measures include:

- Average accumulated swarm fitness (f_c) is the average swarm fitness during the last generations. This metric reflects the performance of the swarm at the end of the evolution. In our experiments, we compute the average over the last *budget* generations.

- Fixed-budget swarm fitness (f_b) is the swarm fitness reached at a certain generation (computational budget). This measure helps to compare different methods on the same grounds, *i.e.* using the same amount of computational resources. In our experiments, we measure this value at $100\% - budget$ of the evolution, which corresponds to the first generation considered in the computation of f_c.

- Time to reach target (g_f) is the first generation at which the predefined target fitness $target$ is reached. If this level is never reached, g_f corresponds to the last generation. This metric

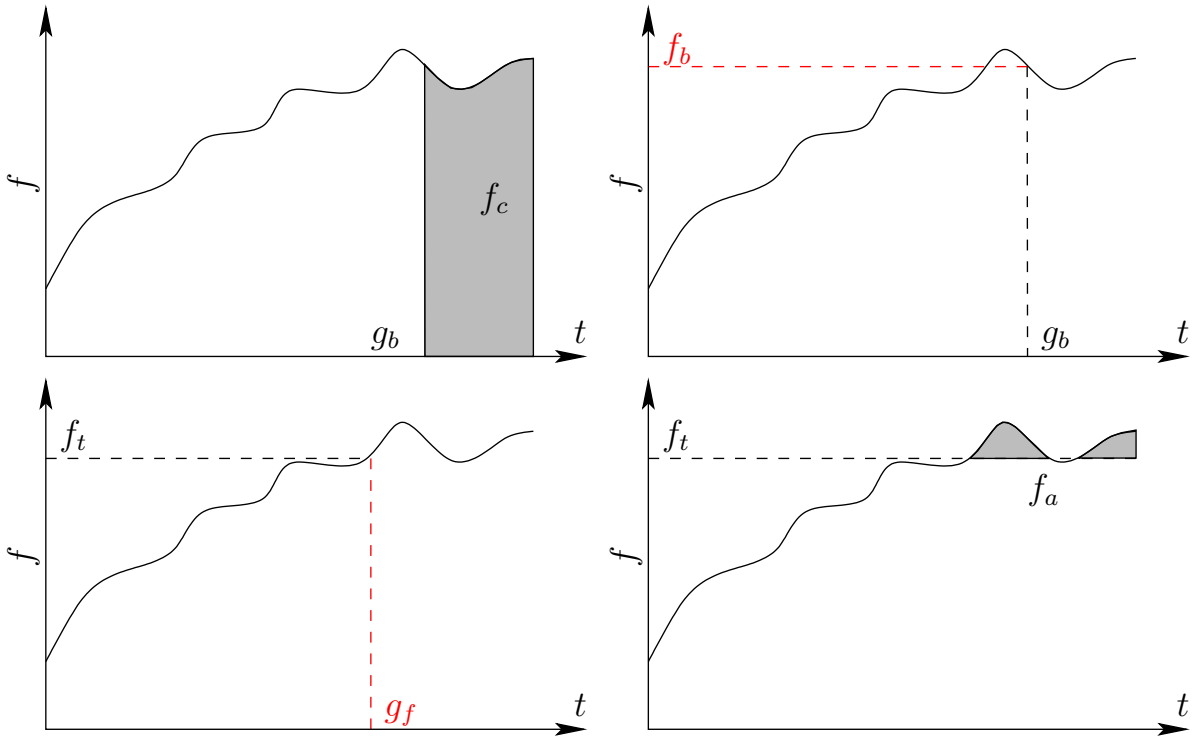

Figure 4.3 – A pictorial description of the four performance measures. From left to right, then top to bottom: average accumulated swarm fitness (f_c), fixed-budget swarm fitness (f_c), time to reach target (g_f), and accumulated fitness above target (f_a).

reflects a certain convergence rate of the algorithms, *i.e.* how fast the swarm hits the target fitness in the task at hand.

- Accumulated fitness above target (f_a) is the sum of all swarm fitness values above the predefined target fitness *target*. It reflects to which extent the target level is exceeded and if this performance is maintained over the long run.

These comparison measures are not to be taken individually. For instance f_c and f_b complement each other and give an indication of the level and stability of the performance reached by the swarm at the end of the evolution. If f_c and f_b are close then performance of the swarm is stable. Also, g_f and f_a combined reflect how fast a given fitness level is reached and to which extent that level is exceeded. Adding the two latter measures to f_c shows if that trend is maintained.

Our experiments on swarm behavior evolution are run in simulation using the Roborobo! simulator [Bredèche *et al.*, 2013]. In the next section, we discuss the use of simulators in our work and describe Roborobo!.

4.1.1 Roborobo! Simulator

A fundamental question when evolving robot control in ER, which is accentuated when using EE approaches, concerns how to evaluate the controllers in the population. Two main options arise: evaluating on real robots or in simulation. There are intrinsic advantages of evaluating on real robots: for example, the *Reality Gap* [Jakobi *et al.*, 1995], which is due to inaccuracies in simulators, is bypassed. In a simulator, such innacuracies are opportunistically exploited by EAs.

Even if some feature in the simulation does not correspond to reality, the EA may use it if it allows for increased performance. This is a strong argument for performing evaluations in real robots. However, conducting experiments on such real robots entails several drawbacks:

Speed: running experiments on real robots cannot be accelerated further than actual time; this is not the case of simulators, which may be sped-up by orders of magnitude, depending on the complexity of the simulation.

Cost: real robots are expensive, and they demand a considerable maintenance and setup effort, and even more so when a swarm of hundreds of robots is used.

Danger: evaluations on real robots may endanger surrounding humans, environmental structure, and the robots themselves.

Flexibility: real-robot experiments "only" allow for the investigation of questions that are physically feasible with current technology; simulators, on the other hand, allow for exploring further questions, including for example morphological robot evolution.

The experiments in this thesis are all run in simulation, using the Roborobo! simulator [Bredèche *et al.*, 2013]. Roborobo! is an open-source, fast and simple robotics simulator, which is intended for large-scale collective robotics experiments. It runs in C++, using the SDL library for fast 2D graphics, and it has been used to simulate up to 4000 robots simultaneously interacting in an environment [Bredèche, 2014]. Roborobo! simulates a swarm of robotic agents that are loosely based on the Khepera or the ePuck robot models, including sets of IR-like proximity sensors and differential-drive movement. It simply manages collisions between robots and with obstacles, and dynamics are minimally represented.

Roborobo! stands in between extremely accurate but slow robot simulators, such as Gazebo or Webots, and fast but minimally functional and inaccurate agent simulators, such as Netlogo. Furthermore, Roborobo! is explicitly intended for simulating robotic swarms that evolve using EE algorithms. It has been extensively used in a number of research contexts in Evolutionary Swarm Robotics, including self-assembly, open-ended environment-driven evolution, navigation, and collective item collection.

4.2 Adaptation to Changing Conditions

We use distributed EE algorithms to learn behaviors for swarms of robots. Our goal is to investigate how a swarm of robotic agents can adapt to unknown and possibly changing environments or tasks. Biological brains are capable of mastering a high number of skills, remembering them even if they were learned a long time ago, and using such skills to address similar problems or combine them to solve a more complex task. For instance, a human baby, in its first months of life, explores the environment around it, by moving and evaluating the outcome of its actions. Based on its findings, the baby acquires basic motor skills, such as moving its arms to grasp nearby objects. At some point, the baby may want to reach an object that is too far away to be grasped without approaching it first, so it needs to learn an additional task: that of moving and approaching the object. It is in this situation that the baby will reuse the previously acquired motor skills to learn how to move and approach the targeted object, for example, by crawling. The baby is able to learn how to crawl because it remembered how to move its arms, and knew how to exploit it, not only for the original task (grasping a nearby object), but also to facilitate the more complex task of approaching a distant object.

Regarding the artificial counterpart of brains, since ANNs store information on a task in the entire set of synaptic weights, when they deal with a sequence of tasks to learn, new learning overwrites and causes the sudden forgetting of any previously learned knowledge. This problem, catastrophic forgetting [Ratcliff, 1990, French, 1999], introduced in Chapter 2, means that an ANN trained on a first task loses such a skill almost instantly when trained on a second task. For individual ANNs to achieve incremental learning capabilities, three underlying requirements need to be solved: ANNs need to be able to perform several tasks (*multitask*), to retain knowledge from previous tasks when learning new ones (*avoid forgetting*), and to reuse previously acquired knowledge to learn subsequent skills (*transfer learning*).

We consider the following scenario for adapting to unknown and possibly changing conditions in a swarm of robots. We use EAs, which progressively improve populations of ANNs. We deploy a robotic swarm in an unknown environment. The swarm uses a distributed EE algorithm to progressively adapt to its environment. At some point, the environmental conditions may change, and the swarm needs to adapt to such a change. The final population of neurocontrollers in the swarm from previous conditions seeds the initial population for the new situation. As such, the swarm needs to continuously adapt to unknown and changing environments. This process iterates in an open-ended manner, while environmental conditions are possibly repeated over time.

In the next section, we present the novel view of the forgetting problem that we consider in our work, which may occur when a swarm of evolving robots adapts to changing conditions in an online manner.

4.2.1 Forgetting Problem

Adapting to changes implies that there is a possibility of previously acquired knowledge being forgotten. In our work, we consider two levels of forgetting: at the individual level and at the populational level. On the one hand, forgetting at the individual level is akin to the problem of catastrophic forgetting as considered in the literature of ANNs in ML. In our case, a swarm is said to forget at the individual level if the neurocontrollers of all or part of the individual robots lose the reached performance before the change, once they start to adapt to a new situation. On the other hand, we propose a view of forgetting at the populational level, *i.e.* at the level of the swarm. In this case, a swarm is said to forget at the populational level if, upon learning of a new task, all the robots in the swarm lose the attained performance that they had on a previous one. Forgetting at the populational level is not the usual manner in which forgetting is considered, although there exist EDO approaches (*cf.* Chapter 2) based on retaining previous well-performing individuals, from which we take inspiration to address forgetting.

The nature of these two levels of forgetting is different. Consequently, forgetting-avoidance mechanisms at the individual or at the populational level should be different. In the case of individual forgetting, the learning algorithm needs to push each controller to keep what was learned before; in the case of populational forgetting, the algorithm needs to promote retention of a set of individuals that performed well in previous tasks in a subset of the population.

4.3 Research Questions and Contributions

The contributions in this thesis aim to shed light on swarm robot behavior adaptation using distributed EER approaches to adapt to unknown environments. Here, we describe the research questions investigated in our work, as well as our contributions to answer them, and how they align toward the adaptation of swarm robot behavior. A first question we ask in our work is:

Does selection pressure play a role when a swarm adapts to given tasks in EER?

In Chapter 5, we investigate the influence of selection pressure in distributed EER, which entails different evolutionary dynamics than classical centralized ER. We experimentally show that selection pressure toward the considered task is indeed necessary when the goal is not mere survival (*cf.* mEDEA, Chapter 3). Furthermore, we show that the stronger the selection pressure, the better the obtained results. This may indicate that a degree of diversity is already maintained by the distributed local populations of each robot.

Our second question relates to the distributed evolution of behaviors for collaborative tasks. An advantage of swarm robotic systems over single robots is that they can potentially address collaborative tasks. However, distributed EE, where robots learn in parallel, makes collaborative behaviors harder to evolve; indeed, individual robots are rewarded for collaborating, but such a collaboration is only beneficial if other individuals are already collaborating. Therefore, our second question is:

Can distributed EER evolve behaviors for an intrinsically collaborative task?

In Chapter 6, we show the feasibility of distributed evolution for an intrinsically collaborative item collection task, a task that cannot be solved by a single robot. We further investigate the characteristics of the behaviors that allow for collaboration, and we show that, while the task includes different types of items, none of them is neglected.

Our third question relates to the reduction of forgetting at the individual level. In this case, a limiting factor is the storing capacity of the representational structure, robot neurocontrollers. To avoid forgetting at the individual level, evolution needs to increase the size of ANNs, which is done by odNEAT, a distributed EA. that augments neural structures. However, odNEAT uses global information (clock timestamps) in one of its components. Therefore, our third question is:

Can neurocontrollers in EER be augmented using only local information?

In Chapter 7, we propose Gene Clocks, a novel fully decentralized mechanism for marking neural innovations when evolving the topology of neurocontrollers in a swarm of robots. In a set of experiments, including two tasks, navigation and an item collection, we show that our method, Gene Clocks, does not deteriorate the results of evolution when compared to using global information.

On the other hand, forgetting can be addressed at the populational level. In this case, our fourth question becomes:

Can forgetting at the populational level be reduced by promoting diversity based on individuals that performed well in the past?

In Chapter 8, we propose Forgetting-Avoidance Evolutionary Algorithm (FAEA), an algorithm that uses phylogenetic information to promote the retention of good past individuals. We test our algorithm in a set of preliminary experiments with centralized neuroevolution, in a sequence of two alternating tasks. We show that our algorithm allows for the retention of high-performing individuals from the past, thus limiting forgetting at the populational level. Although the version of the algorithm that we test is centralized, we argue that, since it does not rely on any global information, it could be translated to a distributed EE setup in a straightforward manner.

5

Influence of Selection Pressure in Distributed EER

Contents

In this chapter, we review evolutionary selection pressures in both centralized and distributed contexts, and we present our experiments to evaluate the influence of the intensity of selection pressure when using a distributed Embodied Evolutionary Robotics (EER) algorithm. When adapting behaviors with EAs, selection operators drive evolution toward fit individuals by controlling the intensity of selection pressure to solve the given task. These operators and their impact on evolutionary dynamics have been extensively studied in offline centralized contexts [Eiben and Smith, 2003]. Here, we study their impact in online distributed contexts, more particularly in EER, where evolutionary dynamics differ from the offline centralized case: selection is performed locally on partial populations, and fitness values on which selection is performed are not reliable, since controllers are evaluated in variable conditions. Several authors have addressed online evolution of robotic agent controllers in different contexts, where they use different local selection mechanisms inducing selection pressure to drive evolution (*cf.* Chapter 3).

The property of local selection in distributed EER algorithms implies that the genomes are carried by different robots are different. The impact on the evolutionary adaptation of the robot swarm of selection over these local populations is studied in this chapter. In our experiments, we compare a range of selection intensities, and measure their impact on the performances when a robot swarm adapts to a food item collection task. Our experiments show that, in distributed

EER, strong selection pressure (*e.g.* elitist operators) can lead to the best performances, even in the absence of explicit mechanisms to ensure diversity. This is opposed to classical centralized ER approaches, in which a lower intensity of selection pressure is preferred, to maintain diversity in the population, necessary for effective search. In EER, internal populations are built based on local communication when robots meet, which depends on the positions of the robots. As such, robot behaviors influence the diversity of their respective local populations. Our results suggest that, in distributed EER, part of the search exploration of the algorithms is delegated to the robot behaviors. Consequently, distributed algorithms may inherently maintain diversity in the disjoint subpopulations on different robots, which can reduce the need of additional diversity-maintaining mechanisms at the EA level.

In this chapter, we begin by presenting the exploration-exploitation dilemma in search problems. We then relate it to selection pressure and diversity in EAs. Subsequently, we review different selection schemes proposed in the context of centralized EAs, as well as in distributed EER. We then present the algorithm that will serve as a testbed for our experiments, along with the selection operators that we compare. Subsequently, we detail our experimental settings and discuss the results. Finally, we close the chapter with some concluding remarks.

5.1 Exploration *vs.* Exploitation Dilemma

When searching for good solutions to a problem, we are confronted with a choice between exploring new possibilities, and exploiting and refining known solutions. Efficiently solving search problems typically requires a trade-off between exploration and exploitation. This boils down to choosing between acquiring further knowledge, and using existing knowledge. If too much emphasis is set on exploring new solutions, the performance of the system may be low; if too much exploitation is applied, the system may prematurely converge, by stagnating in local optima from which it may be unable to escape. This is known as the *exploration versus exploitation dilemma*, and has been investigated in several fields, *e.g.* psychology [Cohen *et al.*, 2007], neuroscience [Laureiro-Martínez *et al.*, 2015], reinforcement learning [Sutton and Barto, 1998], and ecology [Berger-Tal and Avgar, 2012].

Particularly, in Reinforcement Learning (RL), regulating between exploration and exploitation relates to how the search for good agent policies is done. On the one hand, this search includes exploring policies that are more distant in the search space, which may be worse than current ones, in the hope of reaching better areas of the policy landscape. On the other hand, exploitation in RL relates to improving current policies, by focusing on their progressive refinement to reach the respective optima in their vicinity.

In EAs, the exploration-exploitation dilemma relates to the concepts of *diversity* and *selection pressure* [Eiben and Schippers, 1998]. One the one hand, maintaining population diversity is a critical issue in EAs in general, and in ER in particular. Diversity refers to how different the individuals in a population are, and has been proven to be a key factor to improve performance in EAs. Therefore, promoting and maintaining diversity has been an active topic of research since early works on EAs [Mauldin, 1984]. On the other hand, selection pressure in an evolutionary system refers to the factors that influence the survival rate of individuals in the system, *e.g.* parent and survivor selection operators. Selection pressure in EAs usually favors the survival of the fittest, by copying the current best genomes to seed the next population, while worse individuals become extinct. Selection pressure is known to influence the degree of diversity in a population[4]. A high intensity of selection pressure only allows the best individuals to survive,

[4] Selection pressure is not the only source of diversity and exploration in an EA: variation operators, especially

thus reducing diversity to those best individuals in the current population. On the contrary, weaker selection pressure allows less fit individuals to survive, thus maintaining diversity in the population.

There are different methods to exert selection pressure, and thus influence diversity in EAs and ER. In the following sections, we review selection pressure in centralized and distributed ER, and we discuss the particularities of selection pressure in both cases[5].

5.1.1 Selection Pressure in Centralized EAs

One informal characterization of selection pressure is the capacity of the best solutions to conquer the population in the absence of variation. There are several factors that influence selection pressure in an EA; these include the definition of the fitness function, the selection operators (parent and survivor selection), and the structure of the population. The fitness function defines the search landscape and sets the objective to be optimized. The selection operators tune the locality of the search, defining how strict competition is among genomes to be selected for the population of the following generation. The structure of the population defines which genomes compete among each other, *e.g.* all the genomes in the population, or genomes in the same species. Although the definition of the fitness function may not be straightforward, and is an important topic, especially in ER [Nelson *et al.*, 2009], this falls out of the scope of our study. Here, we consider a given fitness function that rewards the accomplishment of a task. Once we have such a fitness function to evaluate candidate solutions, the question that arises is: how do we select the members of the next population?

Traditionally, EAs maintain a single population, on which selection pressure is globally applied[6]. A large number of selection operators and techniques tuning global selection pressure have been proposed and evaluated over the years in centralized EAs [Eiben and Smith, 2003]. Typical parent selection operators include: fitness-proportionate selection (roulette wheel), stochastic universal sampling, K-tournament selection, rank-based selection, and truncation selection. In terms of survivor selection operators, a number of possible options also exist, including: generational replacement (*e.g.* (μ, λ)), partial replacement (*e.g.* $(\mu + \lambda)$), steady-state replacement, elitist replacement, and random replacement. Both parent and survivor selection operators exert different levels of selection pressure: from low intensity, *e.g.* binary tournament or random replacement; to high intensity, *e.g.* truncation selection or elitist replacement.

In centralized EAs, other techniques influencing selection pressure exist [Squillero and Tonda, 2016]. Among others, these include:

- Elitism, which forces the EA to retain the best individual in the population to ensure the monotonicity of the best fitness in the population.
- Niching, *e.g.* fitness sharing, which performs stronger selection between similar individuals, and weaker selection among different individuals.
- Enhancing search with explicit genetic or behavioral novelty and diversity objectives to help searching.
- Competitive coevolution, which leads to varying intensities of selection pressure, depending on the relative performance with respect to the adversaries.

mutation, are mainly concerned with exploring new candidate solutions, thus adding diversity in the population.

[5] In this work, we mainly focus on methods exerting selection pressure toward a task. For a review of auxiliary selection pressures, especially using multiobjective EAs, the reader is referred to [Doncieux and Mouret, 2014].

[6] This is referred to as panmictic populations, in which all the individuals compete with each other.

- Random immigrant approaches, that periodically add new randomly created individuals to the population.

The influence of selection pressure on the results of centralized EAs has been studied in the literature of EAs, which has revealed a number of insights about its influence on evolutionary dynamics. In the next section, we review selection methods operating at the individual level in swarms of robots using distributed EER, which feature non-panmictic local populations.

5.1.2 Selection Pressure in Distributed EER

In this section, we review several online distributed EER algorithms from the perspective of the selection mechanisms that are applied to ensure the desired intensity of selection pressure to drive evolution. The goal of this section is to provide an overview of how selection is applied in distributed EER. We then focus on selection in mEDEA, which is a minimal algorithm in distributed EER, and the basis for the experiments in this chapter.

A common characteristic of online distributed EER algorithms, as described in Chapter 3, is that each robotic agent carries one controller at a time that it executes (the active controller), and locally spreads copies of this controller to other agents. Consequently, agents have only a partial view of the population in the swarm (a local repository). Fitness assignment or evaluation of individual genomes is performed by the agents themselves and is thus noisy, as different agents evaluate their active controllers in different conditions. Selection takes place when the active controller is to be replaced by a new one from the repository.

Probabilistic Gene Transfer Algorithm (PGTA), introduced by [Watson *et al.*, 2002], is commonly cited as the first implementation of a distributed online ER algorithm. This algorithm evolves the weights of fixed-topology neural controllers, and robots exchange parts (genes) of their respective genomes using local broadcasts. The algorithm considers a virtual energy level that reflects the performance of the robot's controller. This energy level increases every time the robots reach an energy source and decreases whenever communication takes place. Furthermore, the rate at which the robots broadcast their genes is proportional to their energy level, and conversely, the rate at which they accept a received gene is inversely proportional to their energy level. In this sense, selection pressure is introduced by fit robots transmitting their genes to unfit ones.

[Silva *et al.*, 2012b] introduce online distributed NeuroEvolution of Augmenting Topologies (odNEAT), an online distributed version of NEAT [Stanley and Miikkulainen, 2002], where each robot has one active genome that is transmitted to nearby agents. Collected genomes from other robots are stored in a local repository within niches of species according to their topological similarities, as in NEAT. Each robot has a virtual energy level that increases when the task is performed correctly and decreases otherwise. This energy level is sampled periodically to measure fitness values, and, whenever this level reaches zero, the active genome is replaced by one in the repository. At this point, a species is selected based on its average fitness value, then a genome is selected within this species using binary tournament. Each robot broadcasts its active genome at a rate proportional to the average fitness of the species it belongs to. This, added to the fact that the active genome is selected from fit niches, exerts selection pressure toward fit individuals.

Embodied Distributed Evolutionary Algorithm (EDEA) [Karafotias *et al.*, 2011], has been applied to different swarm robotics tasks: phototaxis, navigation with obstacle avoidance and collective patrolling. In this algorithm, each robot possesses one genome, whose controller is executed and evaluated on a given task. At each iteration, robots broadcast their genomes alongside with their fitness to other nearby robots with a given probability (fixed parameter). Upon reception, a robot selects a genome from those collected using binary tournament. This genome is then

mutated and recombined (using crossover) with the current active genome with probability $\frac{f(x')}{s_c \times f(x)}$, where $f(x')$ is the fitness of the selected genome, $f(x)$ is the fitness of the robot's current genome and s_c is a parameter controlling the intensity of selection pressure. To ensure an accurate measure of fitness values, robots evaluate their controllers for at least a minimum period of time (maturation age), during which robots neither transmit nor receive other genomes.

With minimal Environment-driven Distributed Evolutionary Adaptation (mEDEA), [Bredèche and Montanier, 2010] address evolutionary adaptation to environmental conditions without a task-driven fitness function. The algorithm takes a genome perspective in which successful genomes are those that spread over the population of robotic agents, and that requires: (1) to maximize mating opportunities and (2) to minimize the risk for the agents (the genomes' vehicles). At every time step, agents execute their respective active controllers and locally broadcast mutated copies of the corresponding genomes. Received genomes (transmitted by other agents) are stored in a local list. At the end of the execution period (*lifetime*), the active genome is replaced with a randomly selected one from the agent's list and the list is emptied. An agent dies if there are no genomes in its list, *i.e.* if it did not meet other agents during its lifetime. This means that the agent stops, and listens for possible incoming communications. The agent remains dead until it receives a genome from another agent passing nearby. The authors show that the number of living agents rises with time and remains at a sustained level. Furthermore, agents develop navigation and obstacle avoidance capabilities that allow them to better spread their genomes. This work shows that environment-driven selection pressure alone can maintain a certain level of adaptation in a swarm of robotic agents.

[Noskov *et al.*, 2013] propose Multi-Objective aNd open-Ended Evolution (MONEE), an extension to mEDEA adding a task-driven pressure, as well as a mechanism, called market, for balancing the distribution of tasks among the population of agents, if several tasks are to be tackled. Their experiments show that MONEE is capable of improving mEDEA's performances in a collective concurrent foraging task, in which agents have to collect items of several kinds. The authors show that the swarm is able to adapt to the environment, as mEDEA ensures, while collecting different kinds of items, *i.e.* optimizing the task-solving behavior. In this context, each type of item is considered as a different subtask. The algorithm uses an explicit fitness function to guide the search toward better performing solutions. The market mechanism, which takes into account the scarcity of items, ensures that agents do not focus on the most frequent kind of items (the easiest subtask), thus neglecting rarer ones. In their paper, the agent's controller is selected using rank-based selection from the agent's list of genomes. The authors argue that when a specific task is to be addressed, a task-driven selection pressure is necessary. This idea is further discussed in the remainder of this chapter.

In the aforementioned works, the authors use different classical selection operators from evolutionary computation in online distributed EER algorithms. It is however not clear if these operators perform in the same fashion as when they are used in an offline non-distributed manner. In an online and distributed context, evolutionary dynamics are different, since selection is performed locally at the agent level and over the individuals whose vehicles had the opportunity to meet. In addition, and this is not inherent to distributed evolution but to many ER contexts, fitness evaluation is intrinsically noisy, as the agents evaluate their controllers in different conditions, which may have a great impact on their performance.

5.2 EER Algorithm and Selection Operators

In the experiments presented in this chapter, we compare the effect of the intensity of task-driven selection pressure, when using a distributed EER algorithm to evolve behavior in a swarm of robotic agents. In this section, we present the algorithm and the variants that we considered in our experiments.

5.2.1 EER Algorithm

In our experiments, we use a variant of the mEDEA algorithm similar to the one described in Chapter 4 (Algorithm 4.1). The main difference with respect to Algorithm 4.1 is that the algorithm alternates between two phases: an evaluation phase, where the robot runs, evaluates and transmits its controller to nearby listening robots; and a listening phase, where the robot does not move and listens to incoming genomes sent by other nearby robots. The evaluation and the listening phases last T_e and T_l timesteps respectively, and take place at different moments for different robots[7]. Since robots are desynchronized, robots in the evaluation phase are able to spread their genomes to other robots that are in the listening phase. If only one common phase took place, with simultaneous broadcast and reception, unfit robots turning on the spot would be able to transmit their controllers to any fitter robot in the vicinity. This separation in two phases is inspired from MONEE [Noskov *et al.*, 2013], where it is argued that this reduces the spread of poorly achieving controllers.

During the listening phase, the robot stops and listens for incoming genomes from nearby passing robots, *i.e.* robots that are in their evaluation phase. At the end of this phase, a robot has a local list of genomes and fitnesses, referred to as local population. The local population also contains the current genome of the robot. This guarantees that all robots always have at least one genome in their respective populations. This situation occurs particularly when a robot is isolated during its listening phase, and does not receive any other genome. Conversely, in mEDEA, isolated robots stay inactive until they receive a genome from another robot passing by.

To load a controller for the next evaluation phase, robots choose a genome from their respective local populations using one of the selection methods discussed in the next section. The selected genome is then mutated and becomes the robot's active controller. Once the next controller is chosen, the list is emptied: therefore selection is performed on a list of genomes that have been collected by the robot during the previous listening phase. Then, the new controller evaluation phase begins. We consider one iteration of the algorithm (evaluation plus listening phase) as one generation.

5.2.2 Selection Operators

The (parent) selection method selects the new genome among the collected ones based on their fitness. This can be done in different manners, depending on the desired intensity of selection pressure. In this chapter, we compare five different intensities of task-driven selection pressure to the local populations. This aims at giving a large span of intensities of selection pressure.

Our selection method is a tournament selection operator parametrized by a selection pressure parameter, $\theta_{sp} \in [0, 1]$. The higher this parameter, the higher the intensity of selection pressure. This method uniformly samples a number of controllers equal to the size of the tournament and

[7] A small random fraction of time is subtracted from T_e so as the evaluation phases of the robots are not synchronized.

selects the one with the highest fitness[8]. The size of the tournament is the fraction of the local population determined by θ_{sp}, *i.e.* $|\mathbf{l}| \cdot \theta_{sp}$.

When $\theta_{sp} = 1$, the best individual in the local population is deterministically selected. When $\theta_{sp} = 0$, fitness values are disregarded, and selection is random in the local population. Between this two extreme values there is a span of intensities of local selection pressure. We chose five values for θ_{sp}, from the lowest to the highest intensity of task-driven selection pressure: 0, 0.25, 0.5, 0.75, and 1. In the next section, we describe our experiments comparing the impact of each one of these intensities in an item collection task, which is a well-studied benchmark in swarm robotics.

5.3 Experiments

In the experiments, a swarm of robotic agents is deployed in a bounded environment containing static obstacles and food items (black lines and blue dots in Figure 5.1). Robots perceive other robots as obstacles. All the robots in the swarm are morphologically homogeneous (as described in Chapter 4), *i.e.* they have the same physical properties, sensors and motors, and only differ in the parameters of their respective controllers. Each robot has 8 obstacle-proximity sensors and 8 food item sensors.

We use a recurrent neural network as the architecture of the robot neurocontrollers (Figure 5.1). The inputs of the network are the activation values of all the sensors and the 2 outputs correspond to the translational and rotational velocities of the robot. The activation function of the output neurons is a hyperbolic tangent, taking values in $[-1, +1]$. Two bias connections (one for each output neuron), as well as 4 recurrent connections (previous speed and previous rotation for both outputs) are added. This setup yields 38 weights in the neurocontroller. The genome of the controller is the vector of these weights. Table 5.1 summarizes the different parameters used in our experiments.

In the task, robots must collect food items present in the environment. An item is collected when a robot passes over it. This collected item is immediatly replaced by another item at a random location. The fitness of a robot controller at generation g is computed as the number of items collected by the robot during the corresponding evaluation phase of g. Furthermore, since we are interested in the performance of the entire swarm, we define the swarm fitness as the sum of the individual fitness of all the robots at each generation:

$$F_s(g) = \sum_{r \in swarm} f_r^g \tag{5.1}$$

Since F_s includes the fitness values of all the robots in the swarm, it corresponds to global information, which is not available to each robot during evolution. Consequently, it cannot be used by the algorithms themselves to guide learning. We use this post-analysis metric to evaluate and compare the results of evolution using different selection methods.

5.3.1 Measures

As discussed in Chapter 4, a characteristic of EER is that robots learn as they are performing the task in an online manner. In this context, the best fitness ever reached by the swarm is not a reliable measure, since it only reflects a level of performance at one point of the evolution. Consequently, based on the swarm fitness, we compute the four metrics for online evolution

[8]Sampling is performed without replacement.

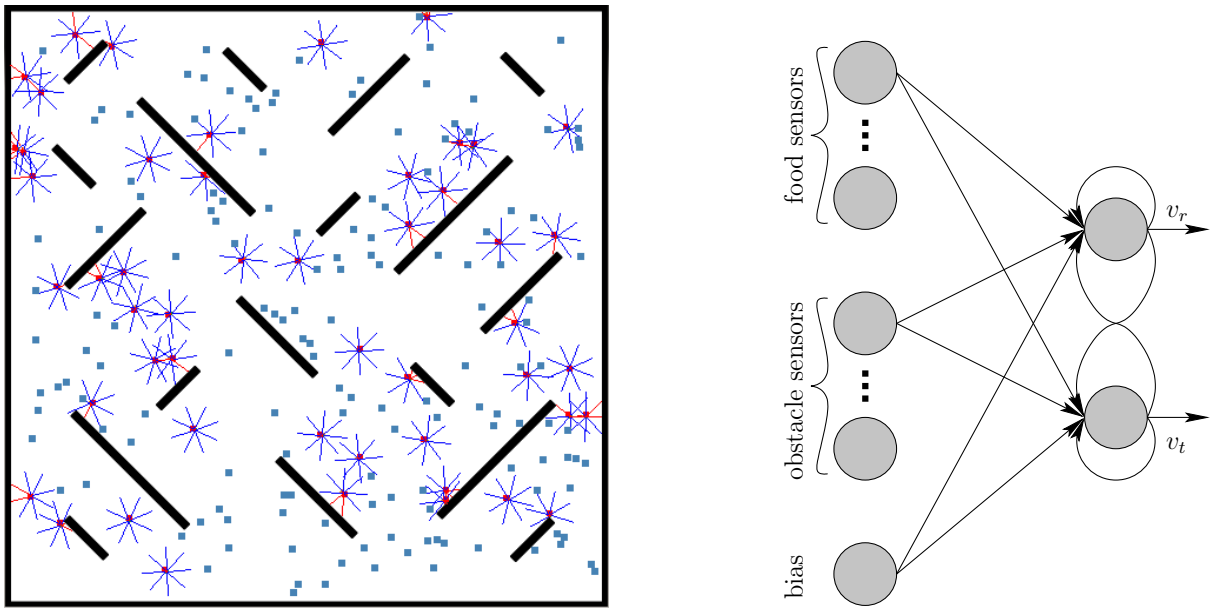

Figure 5.1 – Left: the simulation environment including robots (red dots with thin hairlines representing sensors: blue hairlines if nothing is detected, and red hairlines otherwise), obstacles (dark lines) and food items (blue dots). Right: the architecture of the neurocontroller, a recurrent neural network with no hidden neurons. v_r and v_t are the motor outputs of the robots, corresponding to the rotational and translational velocities, respectively.

described in Chapter 4. These measures include: the average accumulated swarm fitness (f_c) during the last 10% of evolution; the fixed-budget swarm fitness (f_b) at 90% of evolution; the time to reach target (t_f), *i.e.* the first iteration at which a predefined target fitness is reached (75% of the maximum fitness reached over all runs); and the accumulated fitness above target (f_a), *i.e.* the sum of all swarm fitness values during evolution over 75% of the maximum fitness.

5.3.2 Results and Analysis

We perform 30 independent runs for each level of selection pressure, and we measure F_s at each generation in all runs. Figure 5.2 shows the median and interquartile range F_s per generation over the 30 runs for each selection method, based on which we compute the four online performance metrics (Figure 5.3). We performed pairwise[9] Mann-Whitney tests at 95% confidence on these measures between the five selection intensities

Swarm Fitness

The analysis of Figure 5.2 reveals that the swarm rapidly reaches a high fitness level whenever there is a task-driven selection pressure, *i.e.* $\theta_{sp} > 0$. Conversely, without any selection pressure ($\theta_{sp} = 0$, *i.e.* random selection), learning is much slower. Furthermore, for the four former selection methods, the algorithm reaches relatively comparable levels of performance (although slightly lower for $\theta_{sp} = 0.25$, especially when compared to $\theta_{sp} = 1$).

Despite the lower performances achieved by *Random*, the swarm still manages to learn behaviors that collect items. This can be seen in the increasing trend of the swarm fitness in Figure 5.2.

[9]Pairwise in this context means all combinations of pairs of selection methods, ten combinations in our case.

Experiments	
# Items	150
Swarm size	50 robots
Exp. length	5×10^5 sim. steps
Number of runs	30
Evolution	
Evolution length	$\sim$ 250 generations
$\mathbf{T_e}$	$2000 - rand(0, 500)$ sim. steps
$\mathbf{T_l}$	200 sim. steps
Genome size	38
Mutation step-size	$\sigma = 0.5$

Table 5.1 – Experimental settings for the experiments on selection pressure.

As in [Bredèche and Montanier, 2010], even without task-driven selection pressure, the environment drives evolution toward behaviors that maximize mating opportunities, *i.e.* behaviors that explore the environment, and, as a byproduct, they collect items. Robots collect items by chance while they navigate trying to mate. Inspection of the swarm in the simulator reveals that, when selection pressure is present, the evolved behaviors drive the robots toward food items which means that the food sensors are in fact exploited. In other words, evolution drives the controllers to use these sensors. However, without any selection pressure, there is not such a drive, which is also observed in the simulator: robots are not attracted by food items for $\theta_{sp} = 0$.

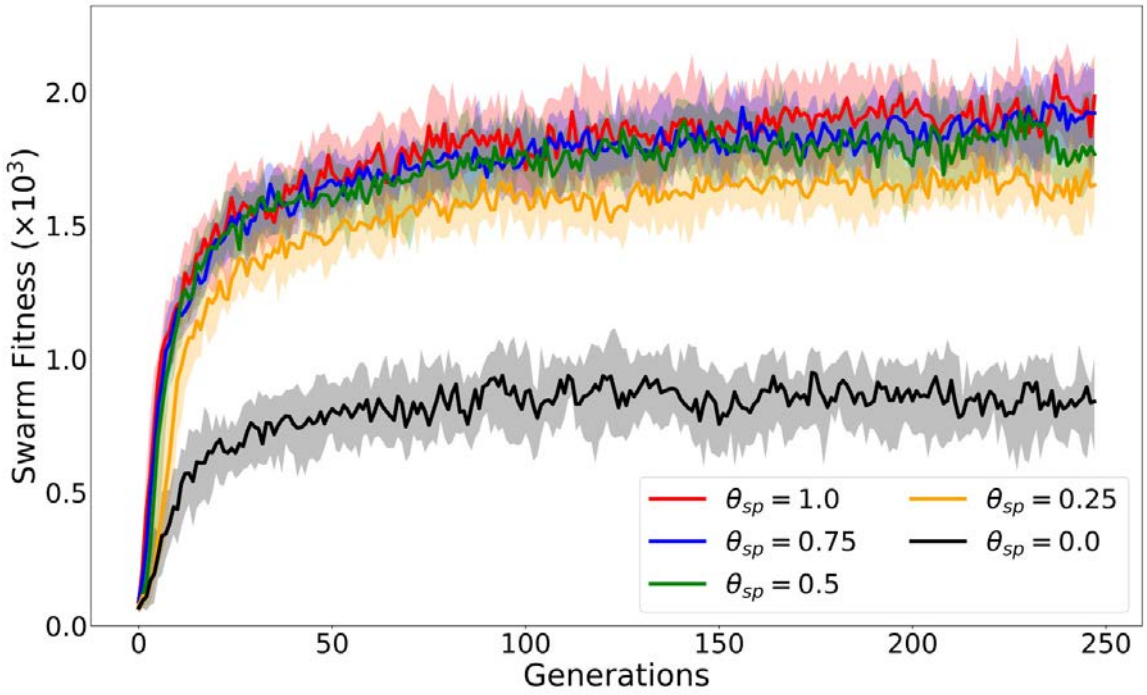

Figure 5.2 – Swarm fitness per generation for the five selection intensities (θ_{sp}) in the item collection task. Full lines depict the median value over 30 runs, while shaded areas represent interquartile ranges.

Online Performance measures

When analyzing the online performance measures introduced earlier, similar trends are observed as for the Swarm Fitness. Figure 5.3 shows the violin plots of the four measures for each selection pressure level over the 30 runs.

In our experiments, there is a significant difference for most of the pairwise comparisons (31 pairwise tests over 40). The tests where there is not such a difference include the comparisons between:

- $\theta_{sp} = 1$ and $\theta_{sp} = 0.75$ for the four measures (p-value > 0.1). This lack of difference seems to indicate that slight differences in high intensities of selection pressure have a very small impact on the performance of the distributed EER algorithm.

- $\theta_{sp} = 1$ and $\theta_{sp} = 0.5$, as well as $\theta_{sp} = 0.75$ and $\theta_{sp} = 0.5$, for the fixed-budget swarm fitness (at 90% of evolution) and for the time to reach target (p-value > 0.05). In these high-medium ranges of selection pressure, a similar performance level for the fixed budget is reached, while it is not consistently maintained by $\theta_{sp} = 0.5$ (see Figure 5.3, top left). Speed of convergence to the target level of performance[10] also seems to be similar, although this level of performance is surpassed by a larger extent by $\theta_{sp} = 1$ and $\theta_{sp} = 0.75$ (see Figure 5.3, bottom right).

- $\theta_{sp} = 0.5$ and $\theta_{sp} = 0.25$ for the accumulated fitness above target (p-value > 0.05). The lack of difference is explained by the fact that not all runs reached the target fitness level for these medium-lower intensities of selection pressure, in which case g_f is the last generation and f_a is almost zero.

Elitist selection with the highest value of selection pressure ($\theta_{sp} = 1$) yields the best overall performance: a high swarm fitness is reached and maintained at the end of evolution (f_b and f_c, Figure 5.3, top right and left). It surpasses the target fitness level in almost all runs faster and to larger extent than the other intensities, especially when compared to low intensities ($\theta_{sp} = 0.25$ and $\theta_{sp} = 0$). Such elitist local selection also manages to reach the required level for all runs (g_f, bottom left), which is not the case of selection pressure intensities below $\theta_{sp} = 0.75$. The target level is surpassed by a generally larger extent than for lower selection intensities (f_a, bottom right). The results show that there is a correlation between the intensity of selection pressure and the performance of the swarm.

Discussion

All task-driven selection pressures yield much better performances when compared to $\theta_{sp} = 0$ (random selection) selection. We have performed a set of experiments with a larger swarm (200 robots), and the results are similar. This is also the case of a set of experiments with a navigation task, where the robots need to maximize movement in a straight line while avoiding static and moving obstacles (walls and other robots). Consequently, selection pressure seems to have a positive impact on performances, when solving a given task, and when the objective is not only to achieve adaptation of the swarm for genome survival, as it was the original motivation of mEDEA. Further, statistical tests show a correlation between the intensity of selection pressure and the performances achieved by the swarm: the stronger the selection pressure, the better the performances reached by the swarm.

[10]The target fitness is 75% of the highest fitness reached by all methods during all runs.

In general, it has been argued that elitist strategies are not desirable in traditional EAs, and the same argument holds for traditional ER. This is due to the fact that such elitist strategies may lead to a premature convergence at local optima. There exists an extensive body of work, especially in non-convex optimization, where it is preferable to explicitly maintain a certain level of diversity in the population to escape local optima, which allows for dealing with the exploration versus exploitation dilemma [Eiben and Schippers, 1998]. This requirement may not be as strong in the context of distributed EER, as our experiments show.

As the local population of a passive robot in its listening phase is built based on the genomes of nearby passing active robots, the robotic agent can be seen as a "vehicle" for the genomes, *i.e.* the algorithm takes a genome-centric perspective. Robots are likely to be scattered enough; therefore a single one does not gather all the genomes of the rest of the swarm during each listening phase. Additionally, this mechanism is embodied in the robot behaviors, *i.e.* which robots gather which genomes depends on the behaviors of the robots. As such, the decentralization of the construction of the local populations in distributed EER maintains a certain level of diversity in the swarm, making explicit mechanisms for maintaining diversity less critical than in classical centralized EAs. Our experiments point in this direction: since the best results in our experiments are obtained with the higher intensity, there seems to be no need for additional diversity-maintenance mechanisms.

There exist optimization approaches, such as spatially-structured EAs [Tomassini, 2005] or island models [Alba and Tomassini, 2002], where distributed subpopulations are evolved. Each subpopulation is evolved by a separate process, and migration mechanisms allow for spread of solutions between processes. These approaches share commonalities with distributed EER, and building bridges with them could help providing further insights on the dynamics of distributed evolution. This could be also the case of Distributed Computer Systems [Coulouris *et al.*, 2011], where distributed algorithms have been thoroughly investigated. For example, gossip algorithms [Shah *et al.*, 2009] are algorithms that spread information through a computer network with a distributed approach, and they have been shown to be robust and efficient. This could be related to genome spread in distributed EER.

5.4 Conclusion

In this chapter, we investigate the impact of different levels of intensity of task-driven selection pressure in distributed EER for swarm behavior adaptation. In these algorithms, selection pressure is applied differently, since each robotic agent has partial views of population, and fitness evaluations are intrinsically noisy due to different evaluation conditions. We compare the performances obtained when using five different intensities of local selection pressure on an item collection task. Our experiments show that local selection pressure at the level of the individual robot systematically improves performances by a large extent, compared to lack of selection pressure (*i.e.* when local selection is random). We also show that the intensity of the selection operator positively correlates with the performances of the swarm on the given tasks, *i.e.* stronger local selection pressure leads to better performances. These results experimentally validate the following statement: when using distributed EER to learn swarm behavior for given tasks, task-driven selection pressure plays an important role, becoming necessary to reach good performances. Strong selection pressure at the local level, *e.g.* elitist operators, *Best* selection, is recommended to further improve performance.

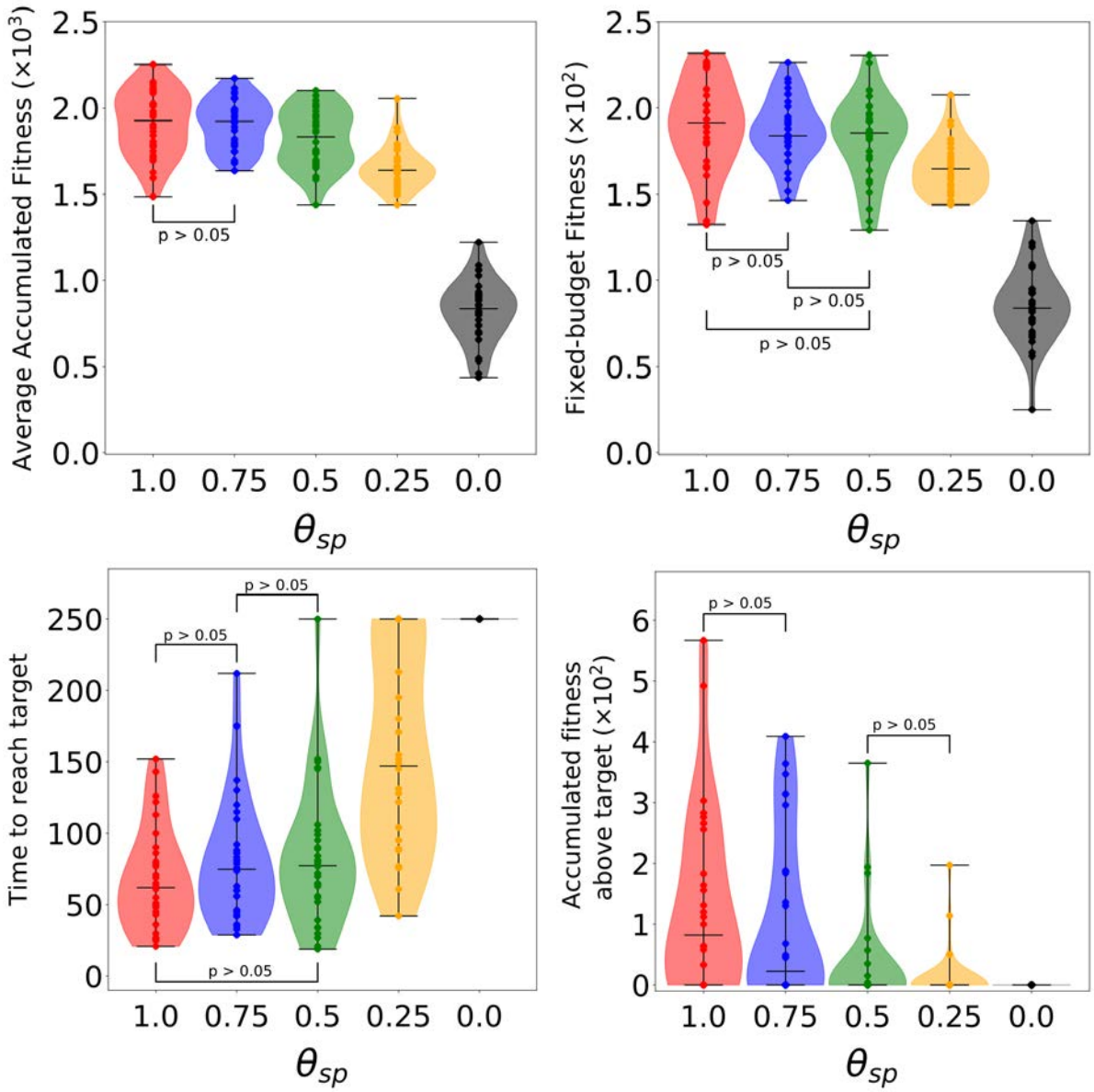

Figure 5.3 – Violin plots showing the kernel density function over 30 independent runs of the comparison measures for the five selection intensities. For each violin plot, the data points are added as a reference. The whiskers show the minimum, median and maximum value. The four plots are, from left to right, then top to bottom: f_c, f_b, g_f and f_a. The label $p > 0.05$ indicates no statistical difference for the corresponding two selection intensities. The rest of the pairwise comparisons yield significant statistical difference at 95% confidence level.

6

Collaborative Item Collection using Embodied Evolution

Contents

In this chapter, we present our experiments on evolving collaborative behaviors using a distributed EER approach. One of the goals of Swarm Robotics is to design agents in a swarm so that they can interact to solve collaborative tasks. How to design such swarm robot behaviors remains a difficult question, for which several methodologies have been proposed [Brambilla *et al.*, 2013,Francesca and Birattari, 2016]. Lacking a well-established engineering methodology to design swarm behavior, machine learning approaches to automatically build swarm robot control systems appear as a promising alternative, and most of the existing approaches belong to the field of Evolutionary Robotics (ER) [Nolfi and Floreano, 2000] and Evolutionary Swarm Robotics (ESR) [Trianni, 2008] (*cf.* Chapter 3).

To perform collaborative tasks, swarms of robots need to coordinate their individual behaviors. The task is then solved by the behavior resulting from robot interactions. Evolving behaviors for intrinsically collaborative tasks is a complex problem. The reason stems from the fact that collaboration is beneficial for an individual only if other individuals also collaborate. Lacking such collaborating partners results in the absence of benefit for an individual to collaborate, which makes cpllaboration difficult to evolve. Several authors have addressed this problem in different contexts [Waibel *et al.*, 2009, Hauert *et al.*, 2014, Bernard *et al.*, 2015]. For example, different genetic team compositions have been studied (*cf.* Section 3.2.1), *e.g.* homogeneous clonal approaches, in which the controllers of all individuals are identical; and heterogeneous approaches, in which each individual carries a different controller. Along another axis, evolution

of collaborative agent behaviors has been studied using different levels at which selection is performed (*cf.* Section 3.2.2): individual level selection, where each agent receives a fitness value depending of its behavior; and team level selection, where the team is assigned a fitness value. In the general case, assigning individual fitness values to each agent is challenging, since their contribution to the collaborative task needs to be estimated. On the other hand, assigning a team fitness value is more straightforward: only an overall evaluation of the completion of the collaborative task is required. In this case, the algorithm is responsible for adequately crediting each agent for their respective contributions to such overall performance.

In this chapter, we study how a fully distributed EE algorithm can adapt swarm robot behaviors for a collaborative item collection task that requires coordination between robots to collect different kinds of food items, which may be seen as coexisting subtasks. The algorithm evolves intrinsically heterogeneous behaviors using individual-level selection, which makes the evolution of collaboration challenging. The contributions in this chapter are twofold:

- First, we show that a distributed EE algorithm run by every robot can efficiently learn a collaborative task where coordination is needed.
- Second, we show that the task is solved by learning a good strategy to coordinate behaviors, instead of learning simple opportunistic strategies. A relative balance between the subtasks is achieved: all kinds of food items are collected, although in different proportions, without resorting to any explicit mechanism to avoid neglecting any of them.

6.1 Evolving Collaborative Behavior

A number of contributions have been made in evolving collaborative behaviors for robot swarms [Nitschke, 2005, Waibel *et al.*, 2009]. This is a compelling problem in the field of evolutionary collective robotics, which has been widely studied with different aims. Most of these works use a clonal approach where all the robots carry a copy of the same controller, leading to homogeneous team compositions [Francesca and Birattari, 2016, Waibel *et al.*, 2009, Tuci *et al.*, 2008, Ferrante *et al.*, 2015]. The teams are evaluated on the global fitness of the group that a centralized EA uses to optimize their behavior, *i.e.* team level selection. On the other hand, there are some works that used cooperative coevolutionary approaches, where the population is decomposed in isolated subpopulations, possibly one per robot [Gomes *et al.*, 2015, Nitschke *et al.*, 2012, Bernard *et al.*, 2015]. Each genome in a subpopulation is evaluated against genomes of the other subpopulations, and evolution proceeds based on such fitness values. In this case, the composition of the team is by definition heterogeneous, *i.e.* the agents carry different genomes.

In most of these works with both clonal and coevolutionary approaches, selection is based on a global fitness measuring the performance of the entire swarm. Further, evolution is performed in a centralized offline manner, where an EA uses these fitness evaluations over all the agents to select the offspring for the next generations. Such team level selection has been shown to favor the evolution of collaborative behaviors, while individual level selection, especially with heterogeneous behaviors, provides challenging conditions for collaboration to evolve. This stems from the fact that collaborating behaviors need to evolve simultaneously to provide a benefit in terms of fitness for different individuals. Indeed, collaboration is beneficial only if other individuals are collaborating. Consequently, collaboration is difficult to evolve in heterogeneous swarms where selection operates at the individual level, since it requires collaborative behaviors to be selected in different individuals simultaneously. Here, we investigate the evolution of collaborative behaviors using a

distributed Embodied Evolution (EE) approach, where behaviors are intrinsically heterogeneous, and selection is applied at the individual level of each robot.

There exist several works that study similar questions regarding the evolution of collaborative behaviors in EE. For example, [Montanier *et al.*, 2016] investigates the conditions for evolving specialization using Embodied Evolution approaches. They concluded that behavioral specialization is difficult to achieve in EE approaches, unless there is some degree of reproductive isolation in the swarm. Additionally, the authors insist on the importance of the size of the swarm and the selection pressure as parameters that may influence the emergence of specialized behaviors. In [Trueba *et al.*, 2013], the authors study specialization of behaviors in robotic collective tasks using a distributed EE algorithm. They provide a theoretical analysis to establish a set of canonical parameters in their distributed algorithm that play a role in the evolution of collective behaviors. Using a swarm of real robots, they further study the impact of each one of these parameters, to conclude that two of them have a great relevance with respect to task performance, namely, the exploitation-exploration ratio, and the replacement probability. In [Haasdijk *et al.*, 2014], the authors proposed a "market" mechanism that explicitly balances between coexisting subtasks, to avoid neglecting hard subtasks over easier ones (*e.g.* the most frequent or the most rewarding). They test their approach in a foraging context where items of different types need to be collected individually by the robots. The authors consider collecting each type of item as a different subtask. Different types of items available in different amount and in different areas of the environment, thus making some of the subtasks easier than others.

One question that we could ask is: how can we evolve behaviors for a collaborative item collection task using distributed Embodied Evolutionary Robotics? In this chapter, we use a simple distributed EE algorithm to evolve swarm robot behaviors for such a collaborative item collection task that includes items of different colors. The algorithm is is a variant of minimal Environment-driven Distributed Evolutionary Adaptation (mEDEA) [Bredèche and Montanier, 2010] that adds task-driven selection pressure at the individual level. We compare the results with a case in which there is no selection pressure toward collecting items. Further, we evaluate the ability of the evolved strategies to collect items of different colors, and conclude that there is no color neglected. It should be noted that this is done without resorting to explicit balancing mechanisms as in [Haasdijk *et al.*, 2014], although items of some colors are collected more frequently than others.

6.2 Collaborative Item Collection

We define a task in which the robots must learn to collaborate to collect these items, since each item needs at least two robots next to it to be collected. The task is a collaborative version of the *concurrent foraging* problem [Jones and Matarić, 2003], a problem in which different kinds of food items are available at the same time and have to be gathered in different ways, rather than having a single resource. In our case, whenever two or more robots are next to a food item and display a signal matching the item, these robots collect the item, one point of reward is split among them, and another item of the same color appears at a random position in the environment. As such, the total number of items and the number of items of each color are kept constant.

The robots in the swarm are initially deployed at random positions in an enclosed circular environment containing food items of different colors. To collect an item, at least two robots must simultaneously reach an item, and display a color signal using an additional colored led effector that is controlled by the robot neurocontrollers. Furthermore, the color signal that the robots display must match the color of the item to be collected. This imposes a synchronization

constraint to the task, so robots are required to have some degree of coordination, or at least to reach a consensus on a color to use when collecting.

All the robots have the same morphology, sensors, and actuators (*cf.* Chapter 4). The sensors include 8 obstacle proximity sensors, 8 robot proximity sensors, 8 food item proximity sensors, and 8 color sensors, each returning a value in $[-1, 1]$ that corresponds to the color of the detected item, if any. Regarding actuators, the robots move using two differential wheels, and have an additional actuator that selects the color to display by the robot, used when collecting.

The controller of each robot is an ANN without hidden neurons. The values from sensors and a bias unit correspond to input neurons, which are fully connected to three outputs neurons for the two wheels and the color effector. Additionally, all the outputs have recurrent connections to the previous right and left wheel speeds. The weights of the neural network that controls both the movement and the color of the led on the robots are subject to evolution using a distributed EE algorithm run by each robot. The ANN computes the weighted sum of the sensors using a vector of synaptic weights that is encoded in the genome, and the activation is squashed using a $tanh(\cdot)$ function taking values between -1 and 1.

6.2.1 Embodied Evolution Algorithm

All the robots in the swarm run an instance of the same distributed EE algorithm, as described in Chapter 4 (Algorithm 4.1), a variant of mEDEA that adds task-driven selection pressure.

This selection pressure is added using an elitist method, dubbed *Best*, which deterministically selects the genome with the highest fitness in the local list. In Chapter 5, we have shown that such a high intensity of selection pressure leads to the best performing behaviors. To provide quantitative comparisons, we also run a variant where selection is done randomly, as in mEDEA, thus disregarding any task objective, *e.g.* collecting items. This provides a baseline for our experiments.

6.3 Experiments

In this section, we describe the experimental settings in our work, as well as the measures and experimental methodology for the post-analysis of the corresponding results.

6.3.1 Settings

In our experiments, 200 robotic agents are deployed in a circular environment containing 100 food items of 8 different colors, with the same proportion of each color. Each robot runs a copy of the algorithm presented in the previous section.

The initial active genome of each robot is initialized with random weights between -1 and 1. When the evaluation period (lasting $T_e = 800$ timesteps) is finished, the robot selects a genome from its local list and mutates it by adding a normal random variable to each gene, with mean 0 and variance σ^2, $\mathcal{N}(0, \sigma^2)$ (in our experiments, $\sigma = 0.1$). Then, the local list of genomes is emptied and a new evaluation phase starts. We consider each evaluation phase as one generation.

When two or more robots are next to an item and they display a color matching the color of the item, 1 point of reward is split among all these robots. Fitness is measured as the sum of rewards obtained when collecting food items. Table 6.1 summarizes our experimental settings. The choice of those is based on preliminary experiments, although the exact values do not change the results significantly. For example, the evaluation period of $T_e = 800$ timesteps is chosen for the robots to have enough time to collect items, the mutation step of $\sigma = 0.1$ is chosen so the

Experiments	
Environment size	1000 × 1000 px
# Items	100
Food item radius	5 px
Swarm size	200 agents
Robot radius	6 px
Sensor range	20 px
Exp. duration	1.6×10^5 sim. steps
Number of runs	30
Evolution	
Termination condition	200 generations
$\mathbf{T_e}$	800 sim. steps
Genome size	105
Mutation step-size	$\sigma = 0.1$

Table 6.1 – Experimental settings in the collaborative item collection experiments.

mutations are not too disruptive, and the density of robots (ratio between number of robots and environment size), as well as the communication and sensor ranges, are chosen to provide enough communication between robots for the distributed algorithm, while not having a too dense environment that would relatively hinder free movement.

We compare our results when using a task-driven selection pressure (*Best* selection) with a variant with random selection over the local list of each robot, as in mEDEA, (*Random* selection), which is our control experiment. Figure 6.1 shows a snapshot of the simulator with a robot swarm and different colored items. For each experiment, we run 30 independent runs to get statistical results.

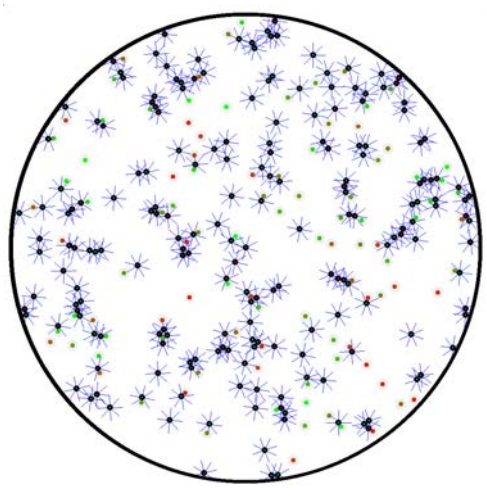

Figure 6.1 – The simulated circular environment containing robots (black circles with thin hairlines representing sensors) and food items (colored dots ranging from red to green in the figure).

6.3.2 Measures

Here, we describe the post-analysis measures used to answer the aforementioned questions, and how we draw conclusions from them. To ascertain if collaborative collecting behaviors are evolved, we measure the total number of collected items by the swarm per generation, that we name *Swarm Fitness*. To provide a reliable measure for online evolution that integrates information over time (*cf.* Chapter 4), we compute the average accumulated Swarm Fitness at the end of evolution as the average Swarm Fitness during the final 20% generations of each run.

Additionally, we measure the average individual reward obtained per collected item, over generations. Since each time an item is gathered one fitness point is split among the robots that participated in collecting it, averaging the individual rewards tells us if items are mainly collected by pairs of robots or by larger groups.

Further, to shed light on how robots collaborate to collect items, we want to evaluate their ability to accomplish the two conditions for items to be collected: simultaneously reaching items, and displaying the right color when close to an item. First, we measure the average ratio of food items that could be collected at any moment over the total number of items, per generation (*i.e.* the proportion of items at every timestep that have at least two robots next to them, averaged for each generation). The better the robots are in reaching items in groups of at least two robots, the higher the ratio of items that could be collected is. Second, we measure at every timestep the average ratio of items that are actually collected among the possible items, averaged for each generation. This gives us an idea of how good behaviors are in terms of synchronizing the color effectors by jointly displaying the right color when collecting. These two measures are indeed two components of the Swarm Fitness.

In order to evaluate if items of all the 8 colors are gathered, we measure the total number of items collected of each color for each run, and we compute the ratio over the total number of collected items (*i.e.* the proportion of collected items of each color in each run). Additionally, we compute the entropy of the proportion of items of each color, which indicates how close is a proportion of items of each color to a uniform distribution where all colors are collected in the same amount:

$$H(\mathbf{p}) = -\sum_{i \in \mathbf{Colors}} p_i \cdot log_2 p_i, \tag{6.1}$$

where $\mathbf{p}$ is the vector of the proportions per color of the total number collected items during each run, and p_i corresponds to each one of the elements of this vector, *i.e.* the proportion of collected items for color i. When all the colors are collected in equal proportion, *i.e.* $\forall i,\ p_i = \frac{1}{8}$, the entropy is maximal with a value of 3. However, when only items of one color are collected, the entropy is minimal with a value of 0.

Finally, we perform pairwise comparisons of the aforementioned color proportions among all 30 runs of each experiment, to test if all the runs of each experiment yield a similar distribution of color proportions. First, the vectors of the 8 color proportions of a run are linearly normalized (by dividing by the Euclidian norm of the vector). Since different runs could evolve a preference toward different colors, we sort the coordinates of the normalized vectors from the most frequent color to the least frequent one. We compute a pairwise similarity measure between each pair of sorted vectors by using the dot product:

$$dot(\mathbf{v}_1, \mathbf{v}_2) = ||\mathbf{v}_1|| \cdot ||\mathbf{v}_2|| \cdot cos(\alpha), \tag{6.2}$$

where α is the angle between the pair of vectors, and $|\cdot|$ is the Euclidian norm. The dot product yields 1 if the two vectors are collinear, 0 if they are orthogonal, and -1 if they are antiparallel. This measure gives us an idea of how similar are runs in terms of proportion of collected items

per color. Note that this similarity measure is computed on the sorted vector, because we are interested in how the runs compare in terms of proportion between colors, not in the actual color value.

To provide statistical results, we show the measures over 30 independent runs. In the case of measures over generations, the plots show the median and the interquartile range of the measure for the 30 runs over time. In the case of single measures, we provide violin plots showing the kernel density function of the dispersion on the data, as well as the datapoints as reference. The violin plots also show the median value, and the whiskers correspond to the maximum and minimum value over the 30 independent runs.

6.4 Results and Analysis

Swarm Fitness

Figure 6.2 shows the Swarm Fitness (*i.e.* the number of collected items) per generation (left) for the experiment with selection pressure (*Best*, in blue) and for the experiment without selection pressure (*Random*, in orange). There is a clear increasing trend showing that the swarm learns how to collaborate to collect items in the case of *Best*. It reaches values of around 150 items per generation. There is also a slight trend of improvement in the case of *Random*, although much lower (around 12 items per generation). This is due to the fact that the robots learn to spread their respective genomes, and, as a byproduct, sometimes two of them meet on an item while displaying the right color, thus collecting the item. On the right, we show the average accumulated Swarm Fitness for both experiments. A Mann-Whitney U test shows that the difference between *Best* and *Random* is highly significant.

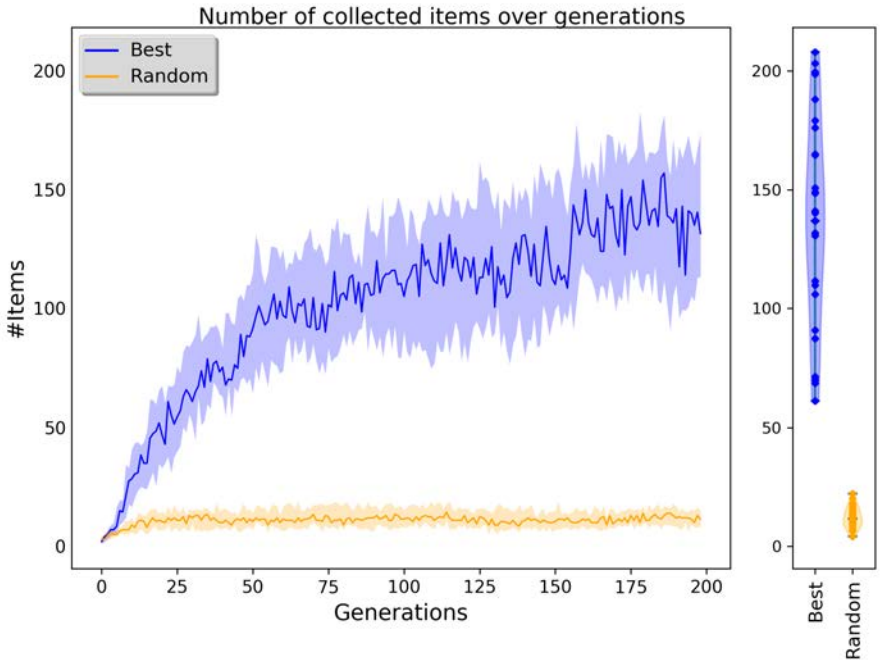

Figure 6.2 – (*Left*) Total number of collected items over time. (*Right*) Average fitness cumulated during the last 20% of the experiments in 30 independent runs. Mann-Whitney U tests show highly significant statistical difference between *Best* and *Random* (p-value $= 1.5 \cdot 10^{-11}$).

The average individual reward obtained per collected item has an almost constant value of 0.5 units in both *Best* and *Random* in all the runs (not shown here). This means that, although collaboration in larger groups is possible, items are almost always collected by pairs of robots, and only very occasionally an item is collected by more than two robots. This is expected, for two reasons. First, reaching items and simultaneously displaying the right color in larger groups (more than two robots) is less likely than reaching it in pairs. Second, when two robots collect an item, they get each a reward of 0.5 units. When an item is collected by more than two robots, the individual reward is lower. Since the number of items is always kept constant, there is no shortage of resources in the environment, and robots do not need to compete for them, *i.e.* they can search for other items to collect rather than collect items in large groups. Concretely, there is a reproductive disadvantage in collecting items in larger groups: robots that collaborate in larger groups do not obtain a larger reward in terms of fitness that would increase their chances of being selected for the following generation.

Efficiency of Collaboration

Figure 6.3 (respectively, Figure 6.4) shows the average at each generation of the ratio of items that could be collected over the total number of items (resp., the average ratio of collected items over the number of items that could be collected). These are two components of the Swarm Fitness, as discussed in the previous section. The trends in the results provide an interesting insight on the evolved behaviors regarding the collaborative item collection task.

The ratio of items that could be collected (Figure 6.3) improves considerably over time when there is selection pressure, reaching values of around 20% of the total items This is a high value, considering the number of items and robots: if an item is collected, robots must search for another one, and thus will spend some time before they are next to it, even with an optimal controller. Consequently, this means that robots evolve very proficient behaviors to find and reach food items.

In the case of *Random* selection, the values are much lower, there is not such an improvement, and the ratio even slightly decreases, stabilizing around 2% of the items in the environment. This shows that, at least in our experiments, reaching items does not increase the chances for spreading robot genomes, and the robots ignore the items. We visually observed that, indeed, the robots in this case do not move toward the items. Additionally, we show on the right the average accumulated ratio of items that could be collected over the last 20% of each run. Mann-Whitney U tests show a highly significant difference between *Best* and *Random*.

The ratio of collected items over the possibly collected items (*i.e.* those that had at least two robots, Figure 6.4) shows a different picture. The measure over generations in the case of *Best* is only slightly higher than for *Random*. We compute the average accumulated ratio over the last 20% of each run (shown on the right). Mann-Whitney U tests show that there is a significant statistical difference between the distribution of values for *Best* and *Random* (p-value = 0.0049), although the effect size of this difference is slim. For *Best*, the value is around 1%, which means that robots only collect around 1% of the items having at least two robots next to them. This directly relates to their overall ability to simultaneously display the right color when they reach a food item, which is not very high (although slightly better than in the case of *Random*).

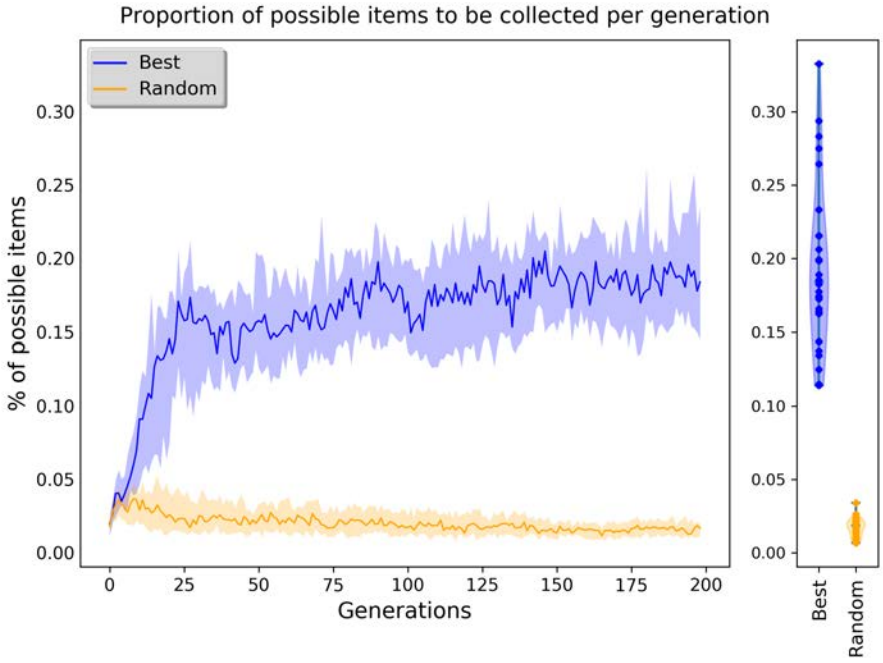

Figure 6.3 – (*Left*) Proportion of items that have at least two robots next to them over time. (*Right*) Average accumulated proportion of items that could be collected during the last 20% of the experiments over 30 runs. Mann-Whitney U tests show highly significant statistical difference between *Best* and *Random* (p-value = $1.5 \cdot 10^{-11}$).

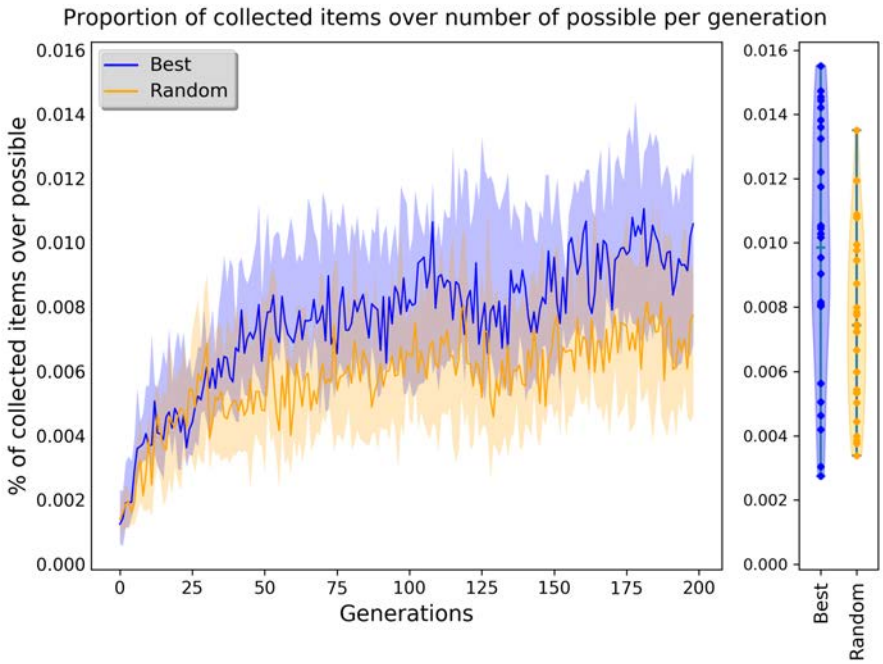

Figure 6.4 – (*Left*) Proportion of items that are collected over the number of items that could be collected over time. (*Right*) Average accumulated proportion of collected items over possible ones. Mann-Whitney U tests show significant statistical difference between *Best* and *Random* (p-value = 0.0049).

To summarize, effective collaborative item collection behaviors are evolved mainly due to the ability of robots to simultaneously reach food items in groups of two robots. However, robots learn suboptimally to jointly display the color matching the item. The cause of this could lie in the encoding of the color output neuron in the neurocontroller, which has a sigmoid ($tanh(\cdot)$) activation function. Such a function squashes the weighted sum of the inputs in a non-linear manner that causes the color output to be biased. Values saturating the neuron toward -1 or 1 are easier to display, because the interval for the weighted sum of the inputs that corresponds to those values is much larger than for intermediate values. This means that the items of the two colors corresponding to the maximum and minimum values are easier for robots to collect.

Color Action Coordination

To further investigate these behaviors, we inspect how items of different colors are collected. We are particularly interested in evaluating if there is a balance in the number collected items for each color. Collecting items of different colors may be seen as different but related subtasks. As previously mentioned, [Haasdijk *et al.*, 2014] proposed a "market" mechanism to avoid neglecting subtasks and balance the effort in a concurrent foraging task using a similar algorithm to ours. In their work, different types of items are found in different proportions, and are detected using separate sets of sensors to emphasize the fact that the subtasks are distinct. In contrast with that work, in our experiments all the types of food items are in the same amount, and their proximity is detected using the same sensors.

Figure 6.5 shows the proportion of the total number of items of each color collected per run in our experiments. The violin plots are grouped in pairs corresponding to *Best* (left) and *Random* (right). The figure shows that, in both experiments, the items of colors in both ends of the range are collected more frequently than the other colors. As said before, this is probably due saturation of the output neuron controlling the color effector of each robot, either toward high or low values. However, there is a significant difference between each pair of proportions (*i.e.* between *Best* and *Random*), as pairwise Mann-Whitney U tests reveal (all p-values < 0.03), except for the proportion of one type of items, orange in Figure 6.5, with a p-value $= 0.1975$.

Furthermore, not only the difference between *Best* and *Random* is significant in almost all cases, but also *Best* systematically yields more balanced proportions, *i.e.* closer to a uniform distribution with $\frac{1}{8}$ for each color. To get quantitative measures of the balance between colors, we show on the right of Figure 6.5 the entropy of the proportion of items per color of *Best* and *Random* (see Equation 6.1). The results clearly show that *Best* leads to swarm behaviors that are more balanced in terms of the color of the collected items than in the case of *Random*. Mann-Whitney U tests show highly significant difference between both experiments in terms of entropy. Additionally, the entropy for *Best* gets close to the maximum value of 3.0, which would correspond to a completely uniform distribution.

In our experiments, without any explicit mechanism as done in [Haasdijk *et al.*, 2014], no item color is neglected. This is probably due to items of different colors being in the same proportion and scattered through the environment, contrary to the aforementioned work. As such, it may seem that collecting items of each type is equally difficult. However, this raises the question of why color proportions are unbalanced in our experiments. A possible explanation is that the colors corresponding to saturated activations (-1 or $+1$) are easier to display. This is due to the encoding of the color effector activation to collect the items (a single value between -1 and 1, discretized into 8 intervals for 8 different colors) and the neural activation function of the controller (a hyperbolic tangent, $tanh(\cdot)$). Consequently, weighted sums of input activations above ~ 1 or below ~ -1 result in a saturated neuron, which corresponds to colors at the ends

of the range.

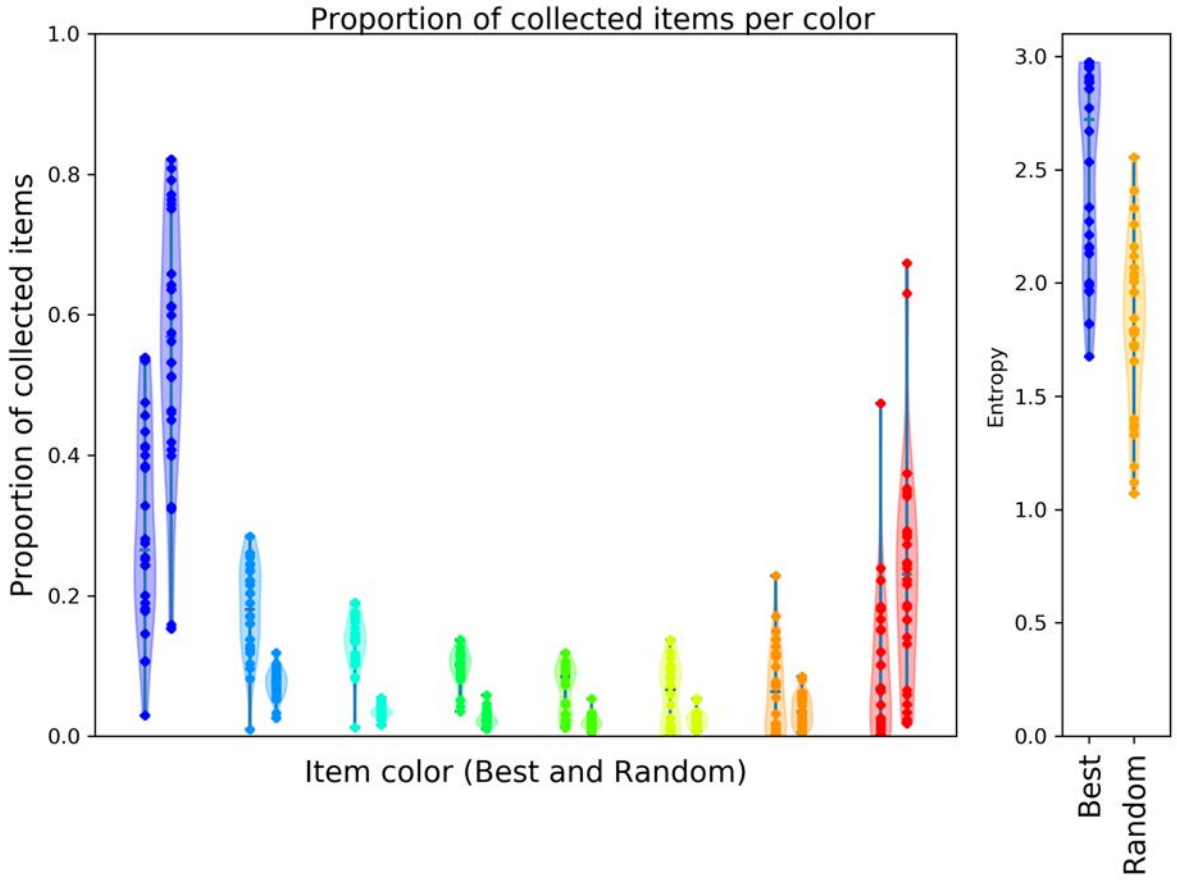

Figure 6.5 – (*Left*) Proportion of collected items of each color over the total number of collected items. Each pair of violin plots shows the proportion of the corresponding color, for *Best* and *Random* selection methods (left and right violin plots of each pair, respectively). (*Right*) Entropy of the proportions of collected items of each color over the total number of collected items, with both selection methods. A Mann-Whitney U test reveals a highly significant statistical difference (p-value = $5.1 \cdot 10^{-7}$).

Finally, we measure the collinearity of the color proportions with the dot product (Equation 6.2) for all the pairwise combinations of the 30 runs of *Best*, as described in the previous section. The results yielded values close to 1.0 (median = 0.946, upper quartile = 0.987, lower quartile = 0.866). This means that the independent runs of *Best* follow a similar trend regarding the balance between colors. Figure 6.6 shows the number of collected items over time of a typical run of *Best*, and the colored areas correspond to the number of items of each color collected per generation. The figure shows that items of all the colors are collected, so no one is neglected, and the number of collected items of each color increases over time, which means that robots progressively adapt to collaborate to collect items of all the colors.

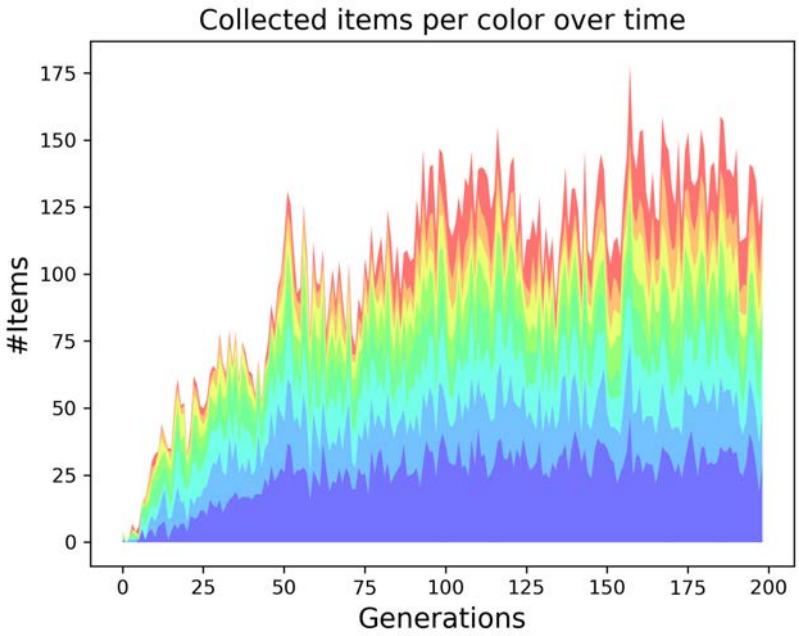

Figure 6.6 – Number of collected items of each color in a typical run of the experiments using *Best* task-driven selection pressure. The top curve shows the total number of collected items, and colored areas represent the number of collected items of each color.

6.5 Conclusion

In this chapter, we investigate the ability of a distributed EE algorithm, mEDEA with task-driven selection pressure, to solve a collaborative item collection task with different kinds of items. The evolution of collaborative behaviors in ER is a challenge that has been widely studied from different points of views and with different approaches. Most of them used either a clonal approach, where the robots display homogeneous behaviors, or a coevolutionary approach, where the robots display heterogeneous behaviors, but subpopulations are set *a priori*.

Here, we use a fully distributed EER algorithm, where each robot runs a separate instance of an EA and they share genetic material when meeting. This intrinsically evolves heterogeneous controllers, and selection operates at the individual level. Such conditions have been shown to be challenging for the evolution of collaborative swarm behaviors. Our work contributes to the question of how to evolve collaboration, by showing that such a distributed EER algorithm is able to evolve behaviors for a collaborative item collection task that requires robots to synchronize and coordinate their actions to collect items of different types. We also show that items are collected by pairs of robots rather than larger groups. Furthermore, we show that the robot swarm evolves behaviors that do not neglect any kind of items, even without an explicit mechanism to enforce it.

Our experiments also show that collaborative item collection is primarily achieved by jointly reaching food items. However, choosing the right color is achieved suboptimally. Further, even if our algorithm evolves behaviors that do not neglect any type of items, the proportions of collected items of each color are not equal. These two issues could be due to the encoding of the color effector in the neurocontroller, which should be further studied.

7

Augmenting Neural Controllers in Embodied Evolution

Contents

In this chapter, we present our experiments on evolving the topology of neurocontrollers for a swarm of robots when using a distributed Embodied Evolutionary Robotics (EER) algorithm. When evolving neural controllers, two main approaches are usually followed (*cf.* Chapter 3). On the one hand, adaptation can operate at the synaptic level, *i.e.* the EA optimizes the weights of a fixed topology neural network. On the other hand, adaptation can also operate at the structural level of the controller, in which case both the topology and the synaptic weights of the neural network are optimized simultaneously. Searching in this richer space allows for the evolution of neural controllers with topologies adapted to the task, and removes the need of hand-tuning such topologies in advance.

There exists a large body of work on the evolution of topologies in neural networks [Gruau, 1993, Stanley and Miikkulainen, 2002, Kowaliw *et al.*, 2014]. However, many algorithms do not consider crossover operators, since such operators, if not designed properly, or if applied blindly, often tend to destruct desirable functional characteristics, or building blocks [Sebag and Schoenauer, 1994]. Among the algorithms that use crossover, NeuroEvolution of Augmenting Topologies (NEAT) [Stanley and Miikkulainen, 2002] introduces a simple gene-marking scheme, called innovation numbers. In this scheme, each element of the neural network, *i.e.* neurons and synapses,

is uniquely identified, and the chronological order of insertion of such elements is tracked, which allows to correctly recombine building blocks.

In the last years, several works have dealt with EER of neural robot controllers in general, usually limiting the evolution to the synaptic weights of fixed-topology neural networks. Recently, online distributed NeuroEvolution of Augmenting Topologies (odNEAT) [Silva *et al.*, 2015] proposed a distributed version of NEAT in the context of EER. The algorithm uses high-resolution timestamps at the robot level to mark structural innovations, such as new neurons or synaptic connections. However, clocks in different robots may drift, thus requiring synchronization, especially if the robots are deployed for long periods of time, which is one of the objectives of EER. In a sense, this requirement is in contradiction with the distributed aspect of Embodied Evolution.

Our contribution in this chapter consists in proposing an algorithm to mark structural innovations that does not rely on a centralized clock or synchronizations. This algorithm, that we name Gene Clocks (GCs), is inspired from logical clock algorithms used in asynchronous distributed system models. By only using local exchanges and with a small computational and communication overhead, Gene Clocks tracks the historical chronology of genes in the population. As such, the distributed nature of Embodied Evolution algorithms is preserved.

In the remainder of this chapter, we start by discussing why the use of topology-evolving neural networks may help adapting to unknown and changing environments Subsequently, we describe different algorithms that address neuroevolution of topologies, both in centralized and in distributed contexts, while discussing how innovations are marked in each case. We then detail our approach to mark neural innovations in a decentralized and distributed manner for a swarm of robots, while explaining its links with asynchronous distributed system models. Subsequently, we describe our experiments to validate our approach, and discuss the corresponding results.

7.1 Motivation

ANNs store information for solving a given task in the entire set of weights. This means that the ANN solves the task by exploiting interactions between all of its elements, *i.e.* neurons and connections. Consequently, there is no single *locus* that represents the task in isolation. Such a distributed representation has several advantageous properties (*cf.* Section 3.1.3): ANNs provide a more compact representation due to intrinsic interactions between their components; information in some parts of an ANN can be exploited as inputs to other parts to solve more complex tasks; and generalization to situations close to those experienced during learning is made possible, since ANNs respond in a similar manner to similar situations.

However, the fact that all the components in an ANN are necessary to solve a task has two major caveats. Firstly, modifying one of such components may have highly deleterious effects in the overall function of the network, which may be completely destroyed. As stated in previous chapters, this is known as *catastrophic forgetting*, and happens when adapting to a new situation on top of previous adaptations. Secondly, the weights of an ANN with a fixed structure have a limited storing capacity, which may become saturated, especially when gradual adaptation to changing conditions is addressed.

Consequently, we are interested in increasing the expressivity and the storage capacity of neurocontrollers, by progressively augmenting their topologies. This aims at allowing for more information coming from new situations to be stored in each neural network, which, in turn, may alleviate the problem of catastrophic forgetting.

Although there exists a body of research in neuroevolution to augment the topology of ANNs in centralized EAs [Kitano, 1990, Gruau, 1993, Angeline *et al.*, 1994, Stanley and Miikkulainen,

2002, Floreano *et al.*, 2008, Mouret and Doncieux, 2009a], approaches that evolve the neural topology in distributed EER settings are scarce, and are reviewed in the next section.

7.2 Evolution of Neural Topologies

In Chapter 3, we have presented an overview of neuroevolution, including the evolution of neural topologies. One issue when evolving neural topologies concerns the *Competing Conventions* problem, which relates to different genetic encodings representing the same neural network. One of the consequences of the competing conventions problem is that crossover operators tend to be disruptive and break existing functional building blocks. NEAT [Stanley and Miikkulainen, 2002] proposes an elegant solution to reduce the effect of the competing conventions problem and perform meaningful crossover. The algorithm marks each new neuron or connection with a unique *innovation number*, which is a historical marker that identifies the new gene and keeps track of the chronology of its appearance in the evolutionary process. These innovation numbers are inherited by offspring networks. The purpose of innovation numbers is to identify the genes expressing the same neural elements in two different genomes, *i.e.* the genes that have the same innovation number, without the need of matching the topologies of the two networks, which is a hard problem [Conte *et al.*, 2004]. As such, the impact of the competing conventions problem is reduced, because genes having the same innovation number in two parent genomes are not repeated when applying crossover. These historical markers are implemented in NEAT by keeping a global counter of innovations and sequentially assigning its values to new genes in the genomes of the population. Furthermore, the algorithm uses a niching mechanism to divide the population in non-overlapping species, based on genotypic similarity, which is also based on the historical markers. To promote topological diversity, NEAT uses intra-species fitness-sharing, *i.e.* networks in the same niche share the same fitness. This results in the protection of new structures that could lead to fitter individuals but require time to optimize their parameters. The algorithm is a centralized offline EA with a global population initialized with minimal fully-connected perceptrons, and where all new genes can be tracked since all information is centralized.

7.2.1 Topology Evolution in Distributed EER

When evolving both the weights and the topologies of ANNs in a distributed EER context, the same problems raised in offline centralized setups (*e.g.* competing conventions, see above) occur. However, a global counter, such as in NEAT's innovation-marking mechanism is not directly transferable to distributed EE, because different robots cannot be simultaneously aware of each other's innovations. Recently, two topology-evolving EE algorithms, and more specifically, two mechanisms to mark topological innovations in a distributed manner, have been proposed, namely IM-(μ + 1) [Schwarzer *et al.*, 2012], and odNEAT [Silva *et al.*, 2015].

In IM-($\mu + 1$) [Schwarzer *et al.*, 2012], each robot draws a random number, between 1 and 1000 in their experiments, to assign it as the identifier of each new neuron gene added in any robot's genome. The authors state that, if the probability of drawing the same number, which is referred to as an identifier collision, is sufficiently low, the system is overall not disturbed, and the impact of colliding innovation numbers is reduced by selection. Although this may be true in relatively short experiments with few robots, when dealing with long-term adaptation EE setups, this probability increases. Additionally, if random identifiers are drawn, historical genealogy cannot be tracked.

On the other hand, online distributed NeuroEvolution of Augmenting Topologies (odNEAT) [Silva *et al.*, 2015] translates NEAT characteristics to a distributed setup. Each local population is

speciated based on a genotypic distance, and the genomes in a species share the same fitness, as in NEAT. Innovation markers are assigned using high-resolution timestamps, and robots mark new genes using their respective local clocks. Offspring are produced by selecting two parent genomes, recombining them as in NEAT, and mutating the result with some probability. Mutations either probabilistically add a new neuron or connection, or add a normally distributed random variable to every weight in the network. The innovation numbers are used to sort the genes in a genome to facilitate the matching of common structures with another genome. As in NEAT, this alignment is used to distinguish between matching and non-matching genes in both genomes. This is then either used to compute a genotypic distance, or to mate two parent genomes without repeating structures. According to the authors, the use of high-resolution timestamps practically guarantees identifier uniqueness, and allows different robots to retain the chronology of the innovations across the distributed system.

Since odNEAT runs on all the robots, which are simultaneously evolving, their respective local clocks have a strong probability of drifting with respect to each other. This is especially true in open-ended long-term adaptation setups, where robots run for a long time and clock drifting may be considerably more important. In this case, innovation timestamps would not be coherent, and thus would not correctly keep track of the actual chronology of innovations. If odNEAT is to be run on physical robots in the long term, it would require a periodical clock synchronization between the robots, which may prove to be impossible when robots are scattered.

Our main motivation for the experiments reported in this chapter, is to propose and experimentally validate a fully decentralized algorithm for topological innovation-marking in neural networks, respecting the distributed nature of EER. To this end, our algorithm, Gene Clocks (GCs), gets inspiration from an event dating mechanism used in asynchronous distributed system models: Logical Clocks (LCs) [Lamport, 1978], a mechanism for ordering messages in distributed systems. By only using local exchanges, and with a extremely small computational and communication overhead, GCs tracks the historical chronology of genes in a distributed EER algorithm.

In the next section, we describe GCs, as well as odNEAT. In our experiments, we compare the results of evolving neural controllers with odNEAT using timestamps, *i.e.* its original marking mechanism, and using GCs as the innovation-marking mechanism.

7.3 odNEAT with Logical Markers

In our proposed innovation-marking algorithm, we see an evolving set of robots in Embodied Evolution as a distributed computer system. In the asynchronous distributed system model [Coulouris *et al.*, 2011], several interconnected processors perform local computations and communicate with each other to solve a common problem. Events in such systems consist of either local computations, or message exchanges. In this work, we consider robots as processors, the evolutionary process of each robot as computational processes, topological innovations as events in the system, and genome exchanges between robots as message exchanges between processors.

One of the fundamental problems in distributed systems concerns dating and ordering events. Since distributed processors do not have a common clock, ordering events with respect to each other requires a specific mechanism. Logical Clock (LC) algorithms [Lamport, 1978] have been proposed to identify and date events in the system in a decentralized way. In this mechanism, every processor has a unique identifier and a monotonically increasing local counter, initialized to zero, to keep track of the events. When an internal event or a message exchange event occurs, the event is marked with the current value of the local counter, which is incremented by one. Whenever a processor receives a message, it updates its local counter with the maximum value

	Innovation identification	Innovation ordering
Gene Clocks	Unique	Total order (using a convention for concurrent events)
Timestamps	Unique in practice	Total order (given no drift)

Table 7.1 – Differences in terms of gene dating mechanism between Gene Clocks and timestamps marking with respect to uniqueness in innovation identification and innovation ordering based in chronology.

between the received counter and its own. This way, events in the system can be ordered with respect to the moment in which they occurred. However, two events may have identical counters if they originated in two different processors and no communication occurred between the two processors during the period in between events. Events that are marked with the same counter are called concurrent events. Whenever this case occurs, concurrent events are ordered by convention, typically with respect to the processor identifier to define a total order between events in the system. It is important to notice that, when there are concurrent events in the system, the order based on processor identifier has no physical meaning, and it is introduced to uniformize the ordering across all processors.

7.3.1 Decentralized Marking of Neural Innovations

In our proposed method, each robot has a unique identifier, r, as well as a sequential *local innovation counter*, c_r, in the same manner as processors have identifiers and event counters in distributed systems. We consider innovation numbers as pairs $< r, c_r >$. Each robot increments its counter c_r by one each time a mutation adds a new gene (neuron or connection), as processors do when they date events. When a robot broadcasts its active genome, all its topological innovations are marked with such innovation numbers.

Using these innovation numbers, genes in a genome can be ordered, since counters keep a chronological sequence of the innovations on each individual robot. However, innovations from different robots cannot be chronologically ordered since all robots maintain independent sequences of innovations. To alleviate this issue, in addition to sending its active genome, a robot broadcasts its current counter c_r, and upon reception, each robot r' updates its counter $c_{r'}$ with the maximum of the received value and their own, as it is done in Logical Clocks (LCs). Concurrent innovations are still possible, *i.e.* two innovations with the same counter c_r, but originating in different robots that do not communicate in between these two events. In this case, innovation numbers are ordered by convention with respect to the robot identifier in the gene, so all robots sort genes in the exact same manner. Table 7.1 summarizes the differences of both dating mechanisms: with timestamps as in standard odNEAT and the proposed decentralized method.

7.3.2 Evolutionary Algorithm

In our experiments, we use odNEAT as the EE algorithm to evolve the topology and weights of neural networks (see Algorithm 7.1). odNEAT follows a similar scheme as Algorithm 4.1 in Chapter 4, and here we describe the main differences between them.

Each robot has an active genome that is initialized with a fully-connected single-layer perceptron, and an internal virtual energy level that represents the performance of the active controller in the task, initialized to a task-dependent default value. Local genome broadcast is done at every timestep with a probability proportional to the genome's fitness. When a broadcast occurs, the robot's active genome, its current energy level, and c_r in the case of GCs, are sent to all neighboring robots. The local population has a limited size and is not emptied at the end of each evaluation, and it contains received and previously active genomes of the robot. When the population reaches its maximum capacity, the addition of a fit genome implies the removal of the worst genome in the population. Given that an internal population is maintained, and that robots locally communicate their active genomes, odNEAT belongs to the class of hybrid EE algorithms, as presented in Chapter 3.

At every control cycle, a robot executes its active controller and updates its energy level according to its behavior. This level serves two purposes. First, it is periodically sampled to estimate a fitness value. This is done to provide a more precise approximation of the actual performance, since the energy level may not be reliable due to varying evaluation conditions. Second, it determines when the evaluation period of the corresponding controller ends, instead of using a fixed evaluation time. This happens when the energy level drops below a given threshold, at which moment a new offspring is produced to replace the active genome, and the energy level is reset to its initial default value. A new genome is given some maturation time T_m to be executed and evaluated even if its energy level drops below the threshold level, as discussed in Chapter 3.

To maintain diversity in local populations, genomes are divided into species based on a genotypic distance, *i.e.* the genomes in the population are grouped in species by genotypic similarity, and fitness sharing is applied in the same manner as in NEAT. Once an evaluation phase ends, the active controller is replaced by a new genome generated as follows. First, a species is selected with a probability proportional to the species fitness. Then, either crossover is probabilistically applied between two parents in the species and the result is probabilistically mutated, or a single parent is selected and probabilistically mutated. Mutation can be of two types: structural mutation and parameter mutation. Structural mutation includes either adding a neuron by splitting a random existing connection, or adding a connection between two random unconnected existing neurons. Regarding parameter mutation, there is a probability of mutating all weights by adding a normal random variable with mean 0 and a standard deviation σ. The mutation procedure is depicted in Algorithm 7.2.

odNEAT also maintains a tabu list to keep track of poor genomes recently encountered. The tabu list is used to filter out received genomes that are similar to one or more individuals in the list, before adding them to the local population. This is done to avoid evaluating candidates similar to known poor solutions. Both the tabu list filtering and the speciation mechanism rely on computing a genotypic distance between two genomes, which is measured as follows:

$$d(g_1, g_2) = \frac{c_1 \cdot E}{N} + \frac{c_2 \cdot D}{N} + c_3 \cdot \overline{W}, \tag{7.1}$$

where $\overline{W}$ is the average of the weight differences between matching genes, *i.e.* genes corresponding to the same structural elements in g_1 and g_2. Non-matching genes are considered in two fashions, depending on the chronology of the compared genes: D is the number of *disjoint* genes, *i.e.* non-matching genes in the middle of the genome, and E is the number of *excess* genes, *i.e.* non-matching genes at the end of the largest genome. N is the number of genes of the largest of both genomes, and c_1, c_2, c_3 are coefficients weighting the relative importance of each of the three elements. The way to distinguish between matching, disjoint and excess genes consists in aligning two genomes using the innovation numbers presented above. Furthermore, the crossover

operator in odNEAT, as in NEAT, also uses historical markers to distinguish between matching, disjoint and excess genes between two parent genomes (see [Stanley and Miikkulainen, 2002] for details).

Algorithm 7.1: odNEAT algorithm run by every robot.

```
1  g_a ← random_genome(), e ← e_init, P ← {g_a}  // And add to a new species
2  while True do
3      if do_broadcast? then
4          send(g_a, e, neighbors)
5      forall the g ∈ received do
6          if Tabu.approves(g) and P.accepts(g) then
7              P ← P ⋃ {g} // And add to matching species
8              adjust_species_fitness()
9      execute(g_a)
10     e ← e + ΔE/Δt
11     if e ≤ e_threshold and ¬ in_maturation_period then
12         Tabu ← Tabu ⋃ {g_a}
13         if random() < p_mate then
14             p_1, p_2 ← select_parents(select_species())
15             g_offsp ← mate(p_1, p_2)
16         else
17             g_offsp ← select_parent(select_species())
18         if random() < p_mutate then
19             g_offsp ← mutate(g_offsp)
20         g_a ← g_offsp, e ← e_init, P ← P ⋃ {g_offsp} // And add to matching species
```

Algorithm 7.2: Mutation of a genome g.

```
1  if random() < p_node then
       // Add a random node to g
2  else
3      if random() < p_conn then
           // Add a random connection to g
4      else
5          if random() < p_w then
               // Add a normal random variable with variance σ
               // to the weights of all connections in g
```

7.4 Experiments

In our experiments, we compare the results of odNEAT when using Gene Clocks and when using timestamps as innovation numbers. Timestamps in a system where local clocks are synchronized

allow for a perfect sorting of genes in a genome, provided that no drift occurs. On the other hand, GCs sorts genes by convention in the case of concurrent innovations. Our main hypothesis is that GCs sufficiently approximate the chronology of innovations to lead to similar results to those obtained using timestamps, while relying exclusively on local information and exchanges. The results of our experiments corroborate this hypothesis, in terms of the quality of learned behaviors, in terms of the size of the architecture of the evolved controllers, and in terms of the number of evolved species.

7.4.1 Experimental Setup

We conducted our experiments on two well-studied tasks in ER [Nolfi and Floreano, 2000], namely navigation with obstacle-avoidance, and item collection. In the navigation with obstacle-avoidance task, robots must learn to move as fast and as straight as possible in a bounded environment, while avoiding both moving and static obstacles, which are other robots and walls. In the item collection task, food items are placed in the environment, and robots must collect as many of them as possible.

In our experiments, a swarm of robots is deployed in a bounded environment containing obstacles (black lines in Figure 7.1), as well as food items in the case of the item collection task (green circles). All robots have the same morphology, as described in Chapter 4. Sensors and motors for each robot include: 8 obstacle proximity sensors, 1 sensor that measures the current energy level of the robot, and, in the case of the item collection task, 8 additional food item proximity sensors. Being morphologically homogeneous, behavior difference between robots originates from having different controllers. The neurocontroller of each robot is initialized with a bias neuron and as many inputs as there are sensors, *i.e.* 9 in the case of navigation, and 17 in the case of item collection. The 2 outputs of the neurocontroller correspond to the right and left wheel velocities. The neurocontrollers are initialized by connecting all inputs to these 2 outputs.

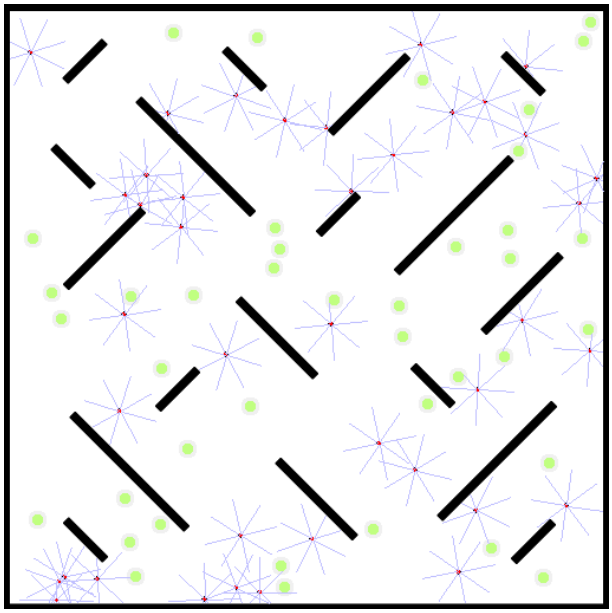

Figure 7.1 – The simulation environment containing robots (red dots with thin hairlines representing sensors), obstacles (dark lines) and food items (green dots).

An active genome life cycle, or genome evaluation, consists in executing it, updating the

energy level, and probabilistically broadcasting it to nearby robots, until such a level is below a task-dependent threshold. When this happens, the genome is considered unfit for the task, and is replaced by a new one, generated as presented in Section 7.3.2. The energy level of the new genome is reset to the default value of 100 units, its maximal value being 200 units. A robot updates its energy level every control cycle differently for navigation and for item collection.

Navigation energy update

The update equation in the navigation task is:

$$\frac{\Delta E}{\Delta t} = f_n(|v_l + v_r| \cdot (1 - \sqrt{|v_l - v_r|}) \cdot (1 - d)), \tag{7.2}$$

where v_l and v_r are the left and right wheel velocities and d is the distance to the closest obstacle. The f_n function is a linear mapping from the interval $[0, 1]$ to $[-1, 1]$. Thus, if a robot moves fast, straight, and far from obstacles it will gain energy, and it will lose energy otherwise. This is the same energy update function in which odNEAT was studied [Silva *et al.*, 2015], and it is inspired from [Nolfi and Floreano, 2000].

Item collection energy update

The update equation in the item collection task is:

$$\frac{\Delta E}{\Delta t} = -0.1 + 10 \cdot c(t), \tag{7.3}$$

where $c(t)$ is a boolean function indicating if an item was collected at t. At each time step, the robot loses 0.1 energy units, and, when it picks up an item, it gains 10 units. In the experiments, 75 items are randomly placed in the environment. When a robot collects an item, it disappears, and a new one is randomly placed in the environment to keep the number of items constant.

Settings and measures

In all the experiments, 100 robots are deployed in the environment. Each robot runs Algorithm 7.1 for 40000 cycles in the navigation task and 60000 cycles in the collection task. We perform 64 independent runs for both innovation-marking methods in both tasks. Every 100 cycles, we measure:

- The average fitness of all robots, or swarm fitness as:

$$F_s(t) \;=\; \frac{1}{|team|} \sum_{r \in team} f_r(t) \tag{7.4}$$

  It represents the overall performance of the evolving robots at instant t.

- The average number of species over the local populations of all robots. The number of species provides information on the overall topological diversity in the swarm.

- The average size of the active controllers of all robots. In our work, we consider the number of connections and neurons as the size of a given neural controller. This provides an indicator of the complexity of the evolved solutions.

We compute the four metrics for online evolution described in Chapter 4, which integrate information over time, since the best fitness reached during evolution is not an adequate indicator of the overall performance of the algorithm. These measures include: the average accumulated swarm fitness (f_c) during the last 10% of evolution; the fixed-budget swarm fitness (f_b) at 90% of evolution; the time to reach target (t_f), *i.e.* the first iteration at which a predefined target fitness is reached (80% of the maximum fitness reached over all runs and both innovation-marking mechanisms); and the accumulated fitness above target (f_a), *i.e.* the sum of all swarm fitness values during evolution over 80% of the maximum fitness.

Taking into account all the variation probabilities in the algorithm (p_{mate}, p_{mutate}, p_{node}, p_{conn}, p_{recur}, p_w), the standard deviation of the weight mutation (σ), the maturation time (T_m), and the local population size, ($|\mathbf{P}|$), we need to set values to 9 parameters (see Section 7.3.2). For each task, the parameters are independently tuned before running the experiments using Iterated Race for Automatic Algorithm Configuration (`irace`) [López-Ibáñez *et al.*, 2011], a parameter-tuning algorithm that optimizes a given quality measure of complete runs of an algorithm. The quality measure used in `irace` is f_c, as presented above. `irace` is a widely used automatic configuration algorithm, that iterates over three phases: (1) sampling new candidate sets of parameters; (2) selecting using racing [Maron and Moore, 1994] the best sets of parameters with respect to the quality measure; and (3) updating the sampling distribution to bias it toward the best parameter sets. These three phases are repeated until an allotted budget is exhausted.

The tuning procedure is performed using timestamps as innovation numbers, and a total budget of 1200 runs is allotted to `irace` to find parameters maximizing f_c, for both tasks. At the end of these procedures, we obtain two sets of parameters, one for each task. All the experiments presented below use these sets of parameters, which are summarized in Table 7.2.

	Tuned		**Fixed**	
	Navigation	**Collection**	**Both tasks**	
$\mathbf{p_{mate}}$	0.842	0.212	$\mathbf{c_1}$	0.5
$\mathbf{p_{mutate}}$	0.154	0.244	$\mathbf{c_2}$	1.5
$\mathbf{p_w}$	0.575	0.559	$\mathbf{c_3}$	0.4
$\mathbf{p_{node}}$	0.057	0.422	**#Robots**	100
$\mathbf{p_{conn}}$	0.184	0.275	**#Items**	75
$\mathbf{p_{recur}}$	0.376	0.526		
σ	0.442	0.114		
$\mathbf{T_m}$	19 cycles	27 cycles		
$\mathbf{\lvert P \rvert}$	5	7		

Table 7.2 – Summary of the tuned and fixed parameters of odNEAT in our experiments, as described in Section 7.3.2.

7.4.2 Results and Analysis

We perform four experiments, *i.e.* two innovation-marking mechanisms in two tasks, in order to test if our proposed method for innovation-marking leads to similar results as the original timestamp-based mechanism of odNEAT. For each experiment, we perform 64 independent runs to provide statistically sound results.

Swarm Fitness

Figure 7.2 (respectively Figure 7.3) shows the swarm fitness F_s, *i.e.* the average fitness of the robots, during evolution, for both innovation-marking algorithms in the navigation task (respectively the item collection task).

For the navigation task (Figure 7.2), both dating methods lead to the learning of proper behaviors, achieving values around the end of the experiments that lay between 75% and 90% of the maximum level of swarm fitness (fixed in the experiments at 200). Upon inspection of some of the evolved behaviors, we have observed that, while having a limited range of perception, the robots are able to rapidly react to other incoming robots, avoid each other, and keep moving straight and fast. As for the comparison between GCs and the timestamp-based dating method, the trend of F_s for both is the same, both in median value and in interquartile range.

Similarly, in the item collection task (Figure 7.3) both experiments manage to evolve controllers that search the environment and efficiently collected items, with performances falling between 70% and 80% of the maximum value around the end of the experiments. When comparing GCs to timestamps, we observe an even clearer overlap between the curves than for navigation. The approximation of the global time with our decentralized method seems to have no impact with respect to the swarm fitness.

To further support this claim, we compute the four aforementioned measures on the 64 runs of each experiment. Violin plots of each measure are presented in Figures 7.4 and 7.5 for each task. We perform two-sided Mann-Whitney tests comparing both dating mechanisms in both tasks, and over all four measures. The p-values of all tests are all above 0.05, *i.e.* there is no statistical difference between the dating methods with respect to any of the measures (see Table 7.3). This confirms our main hypothesis that Gene Clocks sufficiently approximates a perfect innovation ordering to have no impact on the reached fitness level, while being completely decentralized.

We also investigated the size of the architecture of evolved controllers and the size of the species in local populations, to compare both methods in terms of the controllers they produce. The average number of species over time is presented in Figure 7.7 (top), for the navigation task (top left), and for the item collection task (top right). The average size of the neurocontrollers over time, measured as the sum of neurons and connections, is presented in Figure 7.7 (bottom), for the navigation task (bottom left), and for the item collection task (bottom right).

	$\mathbf{f_c}$	$\mathbf{f_b}$	$\mathbf{t_f}$	$\mathbf{f_a}$
Navigation	0.067	0.087	0.13	0.076
Collection	0.349	0.478	0.324	0.307

Table 7.3 – p-values of the Mann-Whitney two-sided tests between the four measures for Gene Clocks and timestamp-based dating mechanisms in both the navigation and the item collection tasks. The differences are not statistically significant for any comparison.

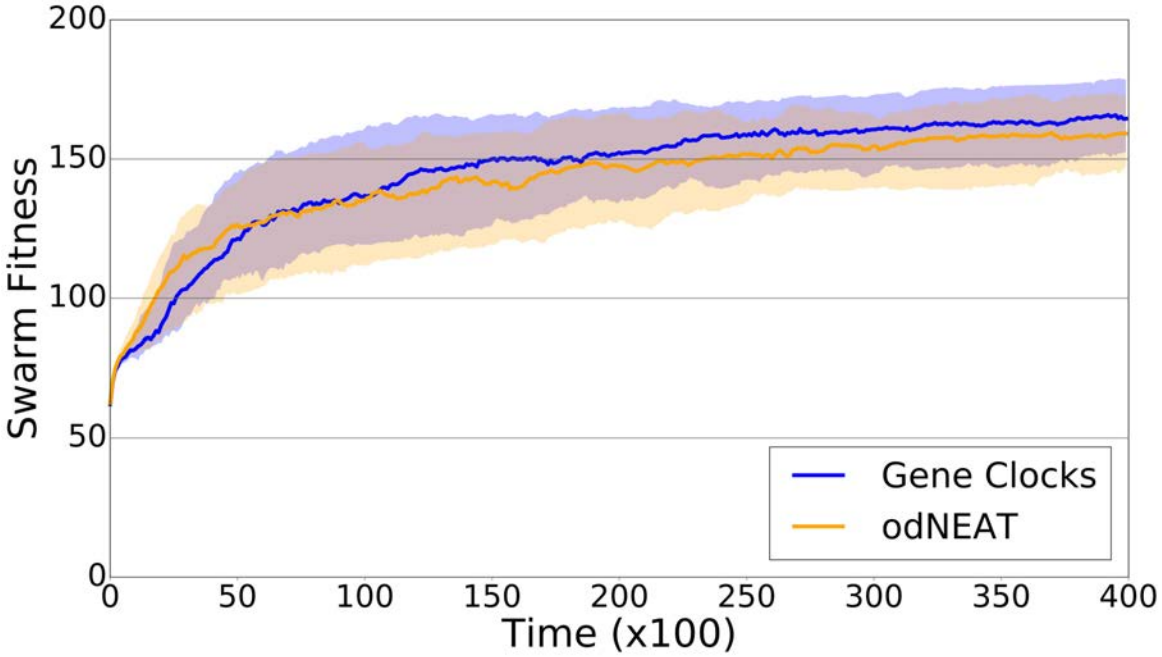

Figure 7.2 – Swarm fitness, F_s, over time for the 64 runs in the navigation and obstacle-avoidance task. The lines correspond to the median value between all runs, and the shaded areas show the interquartile range.

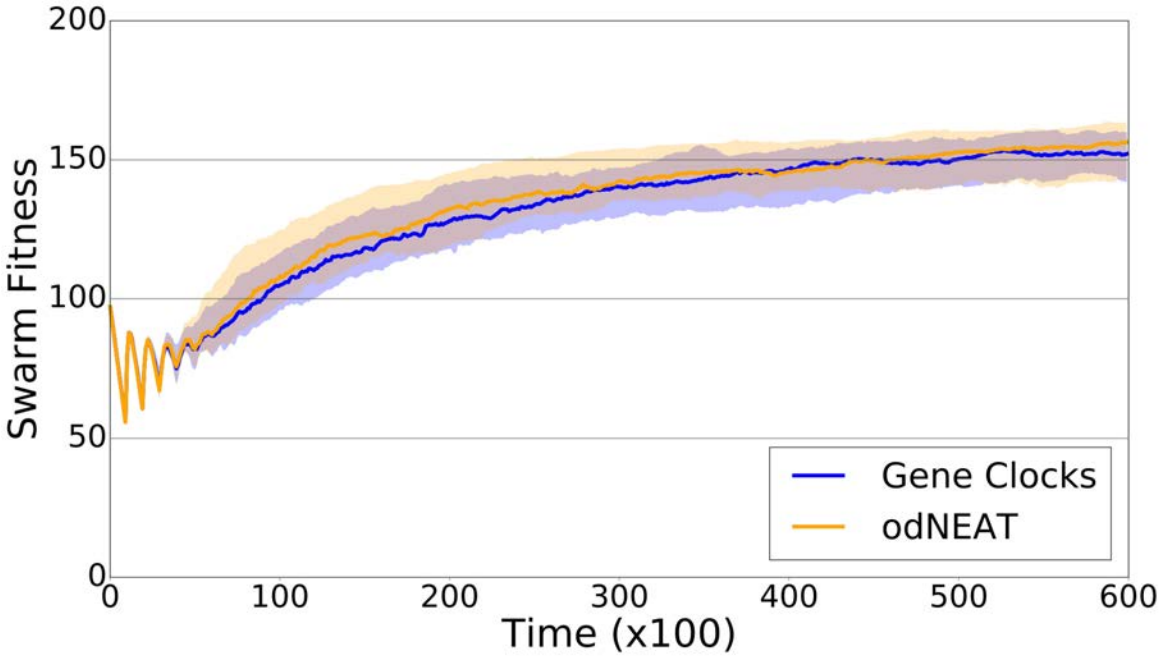

Figure 7.3 – Swarm fitness, F_s, over time for the 64 runs in the item collection task. The lines correspond to the median value between all runs, and the shaded areas show the interquartile range.

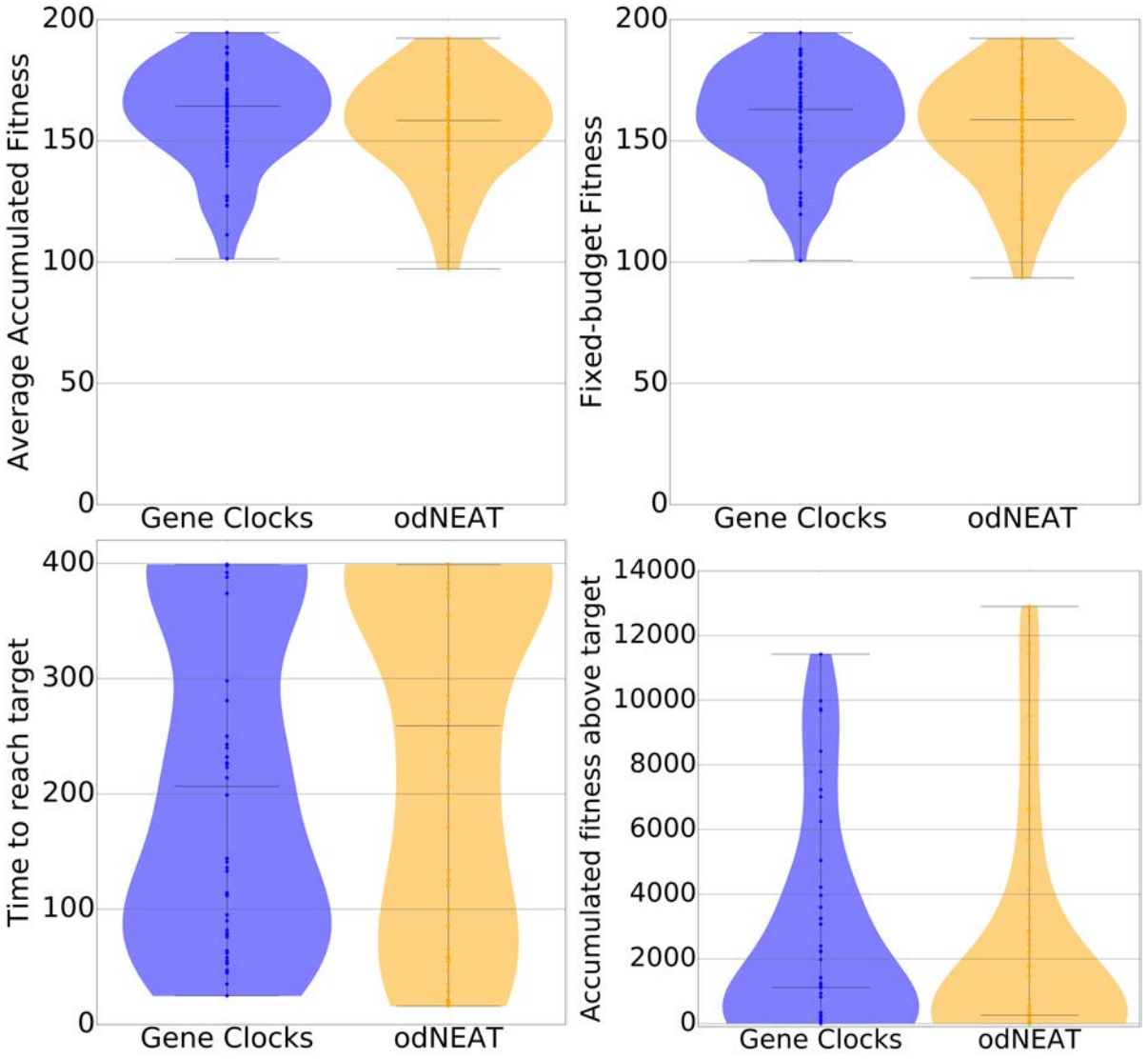

Figure 7.4 – Performance measures of the 64 runs for the navigation task. From left to right, then top to bottom: f_c, f_b, t_f and f_a.

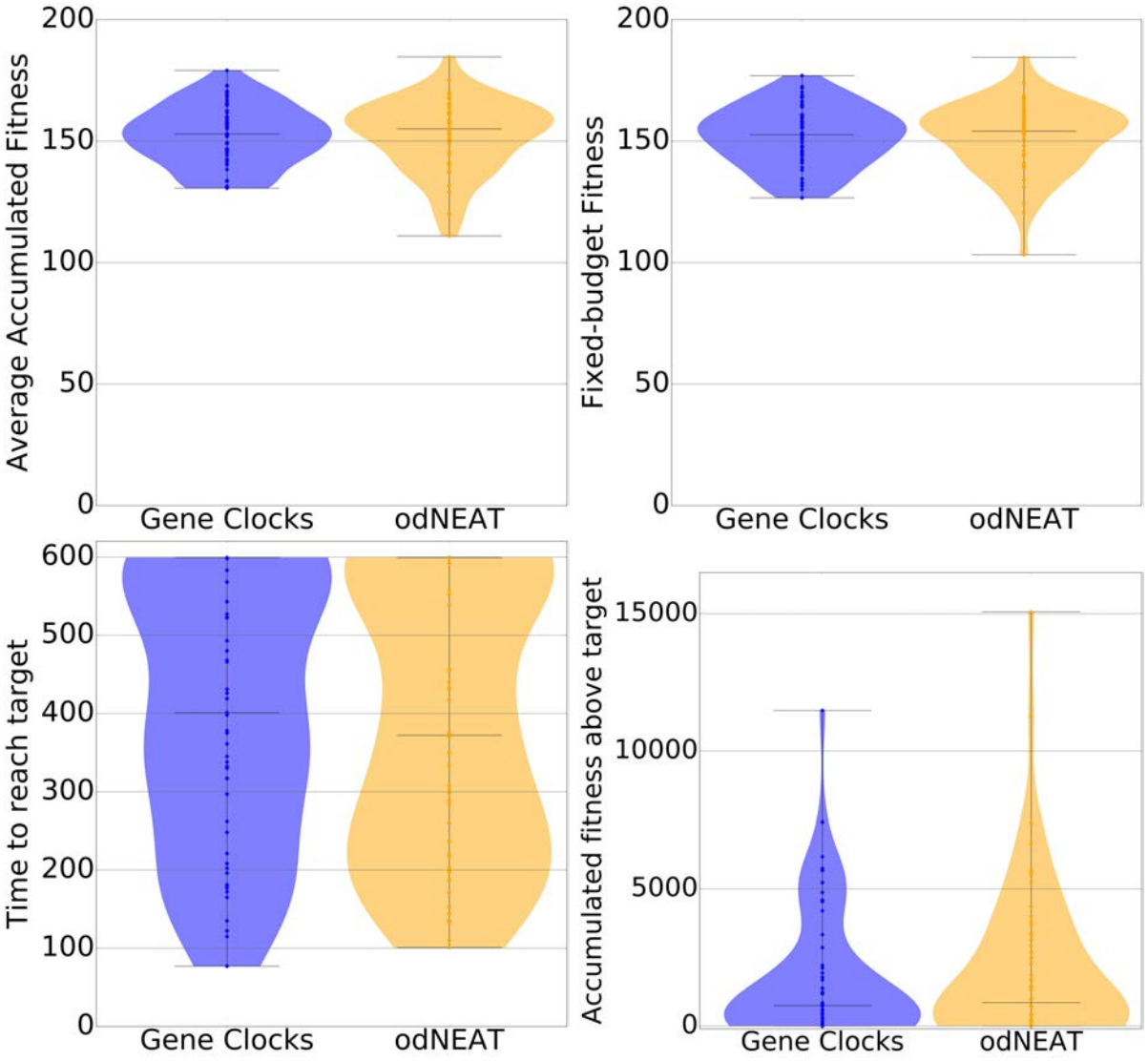

Figure 7.5 – Performance measures of the 64 runs for the item collection task. From left to right, then top to bottom: f_c, f_b, t_f and f_a.

Number of Species

The average number of species per local population follows a similar trend over time in both methods, especially in the navigation task, in which they are almost indistinguishable. As for the item collection task, the median value is the same for both experiments (around 3.75 species per robot), while the variance is slightly larger through evolution for the experiment with timestamps than for GCs (interquartile range of around ± 0.40 species *vs.* ± 0.25 species). Although the difference is slim, the timestamps marking mechanism seems to induce slightly more variability with respect to the number of species. This suggests that the timestamps create more topological diversity in the local populations than our method. We believe that the cause for the disparity is related to innovations being sorted by convention during alignment in GCs. In the experiments, the coefficients used when computing genotypic distance (*cf.* Table 7.2) make excess genes have less impact on genome distance. Given that species are computed based on this value, our marking mechanism may induce a bias toward alignments resulting in more genes at the end of the genome.

Neural Size

Regarding the size of the evolved controllers, the use of GCs creates slightly larger networks of approximately 0.75 more neural elements in both tasks (around 3 ± 1.25 added neural elements *vs.* 2.25 ± 1 for timestamps in the navigation task, and 3.75 ± 1 *vs.* 3 ± 0.8 in the item collection task). We compute the average accumulated size of the neural controllers during the last 20% of evolution (as done for the swarm fitness above). The results in Figure 7.6 show that this difference is not statistically significant in either task (p-values > 0.1). Further investigation is required to ascertain if there is a bias induced by GCs toward slightly larger networks, compared to marking with a global clock shared by all robots.

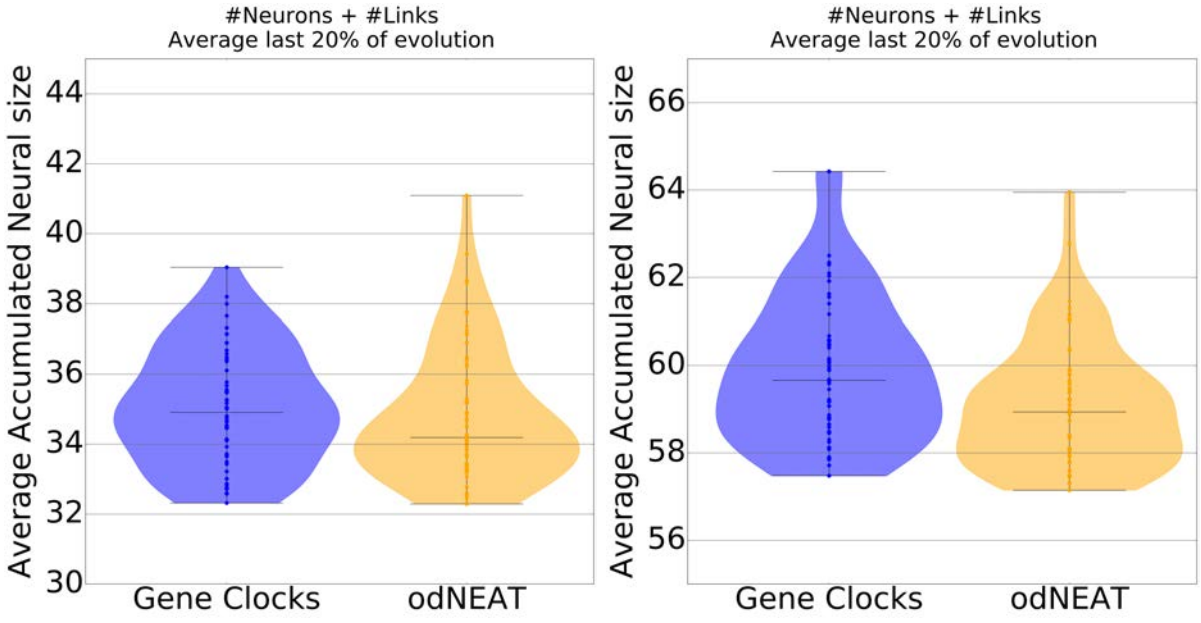

Figure 7.6 – Average accumulated controller size for the navigation task (left) and the item collection task (right).

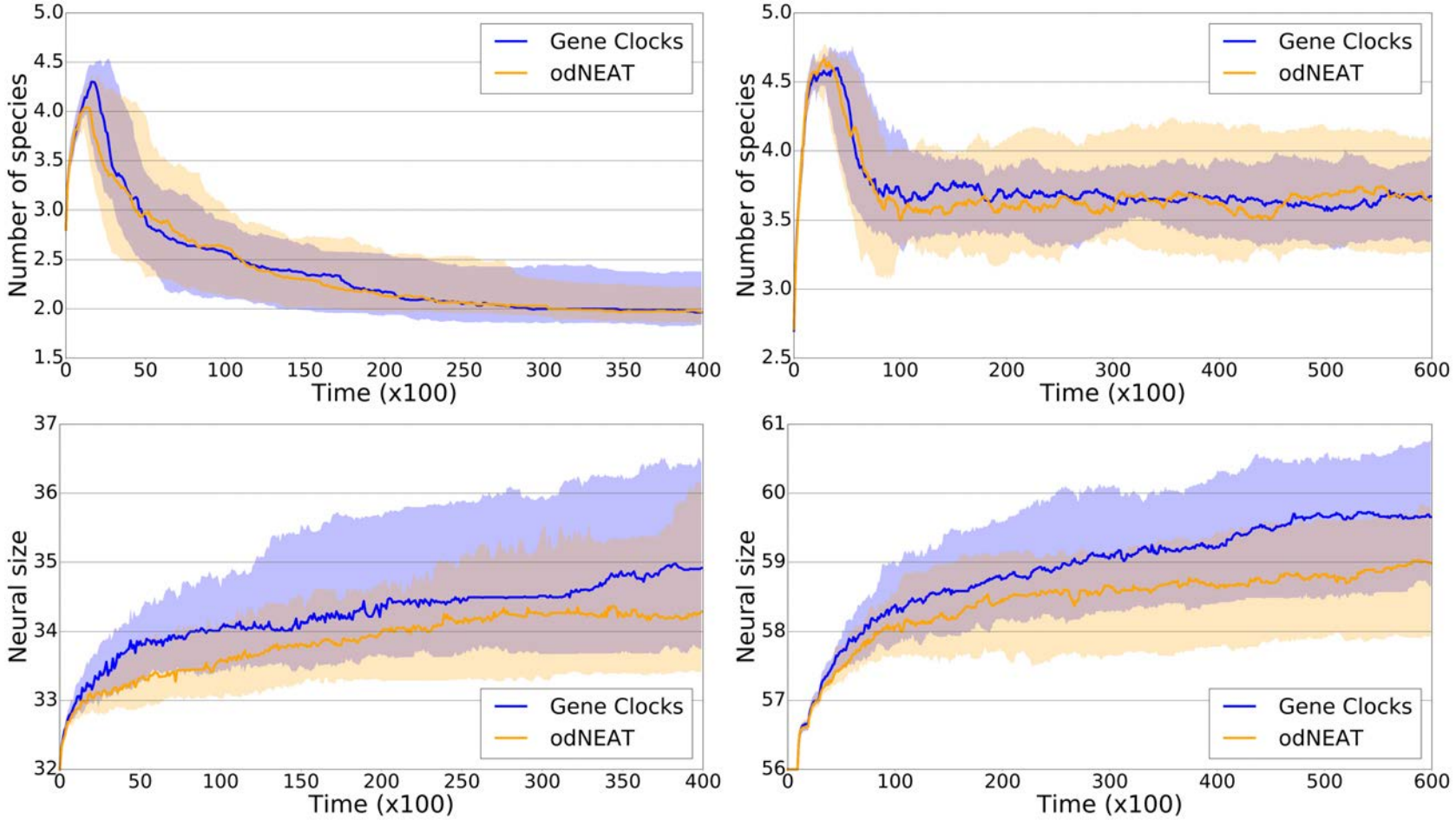

Figure 7.7 – (Top) Mean number of species on all the robots over time, for the navigation task (left), and for the collection task (right). (Bottom) Mean size of the active neural controllers among the robots over time, measured as the sum of neurons and connections of active genomes, for the navigation task (left), and for the collection task (right). The graphics include the results of 64 independent runs.

An example of neural network, evolved around the end of one run of the navigation task with Gene Clocks, is shown in figure 7.8 to illustrate the result of the evolutionary process. Green nodes are inputs, the yellow node is the bias neuron, grey nodes are evolved neurons and the two blue nodes are the right and left wheel speeds.

We have compared our proposed decentralized innovation-marking method with the timestamps mechanism originally proposed for odNEAT in two multirobot tasks, navigation with obstacle-avoidance and item collection. The results validate our proposal of a decentralized innovation-marking method that does not hinder the performance of the timestamp method, and that behaves in the same fashion. Furthermore, our method is completely decentralized, and only requires the addition of one integer to broadcast genomes (the current robot's c_r), a considerably small communication overhead.

7.5 Conclusion

In this chapter, we proposed Gene Clocks (GCs), a novel decentralized mechanism to mark genes with innovation numbers in Embodied Evolution of the topology and weights of neural controllers in a multirobot distributed context. We have described our experiments using odNEAT in two tasks involving a swarm of 100 simulated robots: navigation with obstacle-avoidance, and item collection. We have compared the performances obtained with GCs to a method marking genes with local timestamps, which would require periodical synchronizations to be implemented on real robots. Both methods reach the same level of performances in the considered tasks, and the evolutionary dynamics are similar with respect to the number of species and the size of the controllers. In general terms, GCs innovation numbers approximate the global time, leading to similar performances and dynamics, while being computed in a completely decentralized manner with a low communication and computational overhead. The algorithm presented in this chapter allows for truly decentralized evolution of neural networks with augmenting topologies in Embodied Evolution settings.

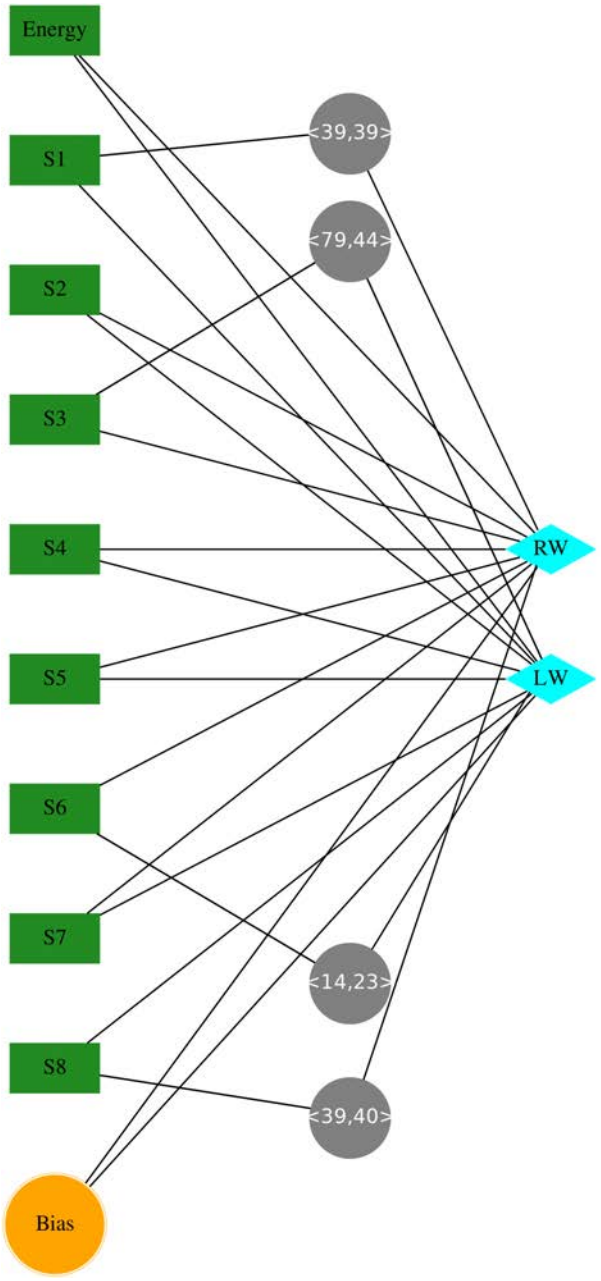

Figure 7.8 – An example of controller evolved for the navigation and obstacle-avoidance task using Gene Clocks. *RW* and *LW* are respectively the right and left wheels of the robot. *S*1 to *S*8 are the obstacle sensors, *Energy* is the energy sensor and *Bias*, the bias neuron. Each added internal neuron is marked with a Gene Clock as a pair <*r*,*c*>. The synaptic weights are removed for readability.

8

Evolutionary Adaptation to Changing Conditions

Contents

In this chapter, we present our approach to avoid forgetting at the level of the population of an EA when adapting to changing conditions. When the environmental conditions change, a learning algorithm needs to adapt online to such changes. The nature of these changes may often be cyclic, where previously seen environments could reappear. If an algorithm that is adapting online in such a changing environment exploits previously acquired knowledge, it will adapt faster to changes when they occur. In the best case, such an algorithm could readapt to previously seen conditions in a transparent manner, *i.e.* by immediately recovering the same level of performance reached in the past.

We start by describing the scenario of adapting to changing conditions, and the problem of forgetting in such a scenario. We then describe our approach to avoid forgetting at the level of the population when using an EA, taking inspiration from a family of Evolutionary Dynamic Optimization (EDO) approaches, which we then discuss. Subsequently, we present Forgetting-Avoidance Evolutionary Algorithm (FAEA), our proposed algorithm to avoid forgetting at the populational level. We test our algorithm in a set of preliminary experiments consisting in an abstract problem of regression, where the goal consists in approximating two alternating 2D functions, showing the feasibility of the approach. The experiments use a centralized version of the algorithm, and we further discuss how to adapt it to the decentralized settings of distributed EER. Finally, we provide conclusions and closing remarks.

8.1 Adaptation to Changing Conditions

The problem addressed by our algorithm consist in adapting agents to changing environmental conditions. Such changing conditions may be seen as changing optimization tasks. These tasks (*e.g.* different fitness functions or different environments) are presented in sequence, one after the other:

$$\mathcal{T} = (t_1, t_2, \ldots, t_{|\mathcal{T}|}), \tag{8.1}$$

where $\mathcal{T}$ is the sequence of tasks, and t_i is the i-th individual task. Thus, agents must first adapt to t_1, and at a certain point, the task changes to t_2, and agents must readapt to this change. In this settings, changes may be cyclic, with past tasks reappearing. If a task $t_i = \alpha$ has been faced by the agents, and, later, the task reappears ($t_j = \alpha$, $j > i$), previously acquired knowledge on the task α could be exploited to better readapt. In the ideal case, this readaptation to a task faced in the past could be instantaneous.

It is known from ML that, when learning in changing conditions, reactive agents with ANN controllers may erase previously acquired knowledge, which is known as catastrophic forgetting (*cf.* Chapter 2). Consequently, mechanisms to avoid such forgetting are needed when adapting to changing conditions. In our work, we argue that such mechanisms to avoid forgetting may be implemented at two different levels, that we call *individual* and *populational* levels to avoid forgetting:

- In the individual-level case, the goal of each learning agent is to incrementally adapt to tasks in sequence, while retaining and progressively integrating new information in its controller. Otherwise stated, a learning agent that faces a sequence of tasks avoids forgetting at the individual level if it is able to solve the tasks faced in the past. For such integration of information to be possible, the controller must allow for storing new knowledge without disturbing previously acquire knowledge.

- In the populational-level case, forgetting is avoided by maintaining previously acquired knowledge spread through a population of agents in the algorithm (*e.g.* candidate solutions in an EA or robots in a swarm). In this case, the population of agents avoids forgetting at the populational level if, for each previously faced task, there is at least one agent that is able to solve it. Thus, diversity needs to be maintained in the population to keep agents that are able to solve each task. In that sense, it is the population that avoids forgetting, and not agents taken individually.

Here, we address the problem of forgetting at the populational level in an EA by allowing individuals that performed well in the past to survive, regardless of their current fitness. We take inspiration from Evolutionary Dynamic Optimization (EDO) [Nguyen *et al.*, 2012], which addresses Dynamic Optimization Problems (DOPs) using EAs. In DOP, the goal is to efficiently and effectively track moving optima through a search space. First, the optima need to be found. Second, when the search landscape changes over time, the search algorithm needs to adapt the population to the change. There exist EDO approaches that maintain diversity in the population during search to cope with these dynamic environments. This consists in saving candidate solutions that performed well in past conditions, even if they are no longer efficient. These individuals are retained to be exploited in the population when those conditions reappear (if changes are cyclic), thus allowing for fast readaptation to previously learned conditions. If such an algorithm faces such a previously learned condition, an individual that performed well in that condition in the past will reproduce faster, thus rapidly attracting part of the population to its region of high performance. For example, with Self-Organizing Scouts [Branke *et al.*, 2000], the authors

propose an EA that maintains diversity by using a large part of the population to search for new optima, while several small subpopulations track changes of each optimum that has already been found. There exist other approaches for DOP that maintain diversity in the algorithm by storing previous good solutions in a separate memory. These solutions are used to replace the worst solutions in the population, either periodically or when a change is detected [Branke, 1999, Yang, 2005, Yu and Suganthan, 2009].

In the work presented in this chapter, our hypothesis is that pushing an EA to maintain individuals with high-performing ancestors in the population helps avoid forgetting at the populational level, thus allowing for fast readaptation to previously seen conditions. Our proposed algorithm has some ties with the aforementioned EDO approaches: it aims at progressively adapting to changing tasks by building a population that keeps well-performing individuals from the past, thus avoiding forgetting. This is done by applying an additional selection pressure toward individuals with high performing ancestors in their lineages. Retaining such individuals, in turn, enhances adaptivity by allowing for an effective and efficient readaptation to previously learned tasks. Our algorithm is described in the next section.

8.2 Forgetting-Avoidance Evolutionary Algorithm

We propose Forgetting-Avoidance Evolutionary Algorithm (FAEA), an EA that aims at avoiding forgetting at the populational level when adapting to a sequence of environments or tasks. The algorithm exploits phylogenetic information on the lineage of ancestors of the current individuals in its population. This means that, at any moment, the goal of the algorithm is twofold:

1. To adapt to the current task with a part of the population.

2. To maintain in part of the population some individuals that were good in the past.

Consequently, two different types of selection pressure are applied. On the one hand, classical selection pressure for the current task using a fitness function is applied. On the other hand, selection pressure toward individuals whose ancestors performed well is also applied. To do so, the algorithm stores information on the phylogeny (*i.e.* the ancestors of genomes in the population) in a data structure that we name the *history*. In the simplest implementation of this algorithm, the history contains, for each genome in the population, the list of fitness values of its ancestors. This can then be used to promote individuals with high-performing lineages, *e.g.* with a high average fitness of the ancestors. Since the fitness values of the ancestors are measured in past generations, they may concern previous tasks: by selecting individuals with high-performing lineages, selection pressure is applied toward possibly different tasks in the past. This is done without requiring any *a priori* knowledge on the number of tasks, neither on which the current task is or on when the task changes.

The algorithm, shown in Algorithm 8.1, is a centralized EA, where a population of candidate solutions evolves over time to adapt to task changes. The algorithm aims at evolving a diverse population that maintains individuals that are good at the current task and individuals whose ancestors were good, possibly in previous different tasks. To do so, the history of lineages is stored in H, which is progressively built to contain information on the ancestors of the individuals in the current population. In our case, for each genome in the population, H contains the list of fitness values of all the ancestors of that genome, which is used when selecting a parent to generate a new offspring. p is the probability of selecting on the basis of lineages, while $1-p$ is the probability of selecting on the current fitness. Thus, p can be seen as a parameter to tune the selection pressure toward one objective or the other.

To generate an offspring with respect to the current fitness, a parent is selected from the current population using current fitness values, then mutated, as it is usually done (*e.g.* with fitness-proportionate selection). To generate an offspring with respect to the history, first a value aggregating the information in the list of fitness of the ancestors of each genome is computed. This provides another objective function that measures the quality of the lineage of each genome. Subsequently, these aggregated fitness values over the different lineages are used to select a parent from the population (*e.g.* using proportionate selection on the aggregated fitness values over the lineages), and it is kept in the new population without modification. Since such an individual is selected using information on evaluations made in previous generations, this mechanism pushes evolution to keep individuals that performed well in the past.

In the next section, we present a set of preliminary experiments, where we test our algorithm in a sequence of two alternating tasks, each consisting in approximating a different 2D function.

Algorithm 8.1: Forgetting-Avoidance Evolutionary Algorithm (FAEA)

Input: N : *population size*, p : *probability of selection on history*
Output: P : *evolved population*

$P \leftarrow initializePopulation(N)$, $H \leftarrow \emptyset$
while $\neg finished$ **do**
  $F \leftarrow evaluate(P)$
  $P' \leftarrow \emptyset$, $H' \leftarrow \emptyset$
  for j *from* 1 **to** N **do**
    if $rand() > p$ **then**
      $(x_j, f_j, h_j) \leftarrow select(P, F, H)$ // *Selection on fitness*
      $o_j \leftarrow mutate(x_j)$
    else
      $(o_j, f_j, h_j) \leftarrow select(P, aggregate(H), H)$ // *Selection on history*
    $h_j.push(f_j)$
    $P' \leftarrow P' \cup \{o_j\}$, $H' \leftarrow H' \cup \{h_j\}$
  $P \leftarrow P'$, $H \leftarrow H'$
return P

Function `aggregate`(*H : histories of the individuals in the population*)
  `// Example of possible aggregate function on history`
  $result \leftarrow \emptyset$
  foreach $h_i \in H$ **do**
    $result \leftarrow result \cup \{avg(h_i)\}$
  return *result*

8.3 Preliminary Experiments

In our experiments, we evaluate FAEA on a sequence of two alternating regression tasks, t_1 and t_2. Each task consists in approximating a 2D non-linear function by providing the y coordinate as output for a given x input coordinate. The functions to be learned (shown in Figure 8.1) correspond to two randomly generated ANNs with 1 input neuron (x), 1 output neuron (y), and 1 bias neuron with a constant value of 1, to which a set of random neurons and connections are

added. Subsequently, the synaptic weights of the resulting networks are randomly modified.

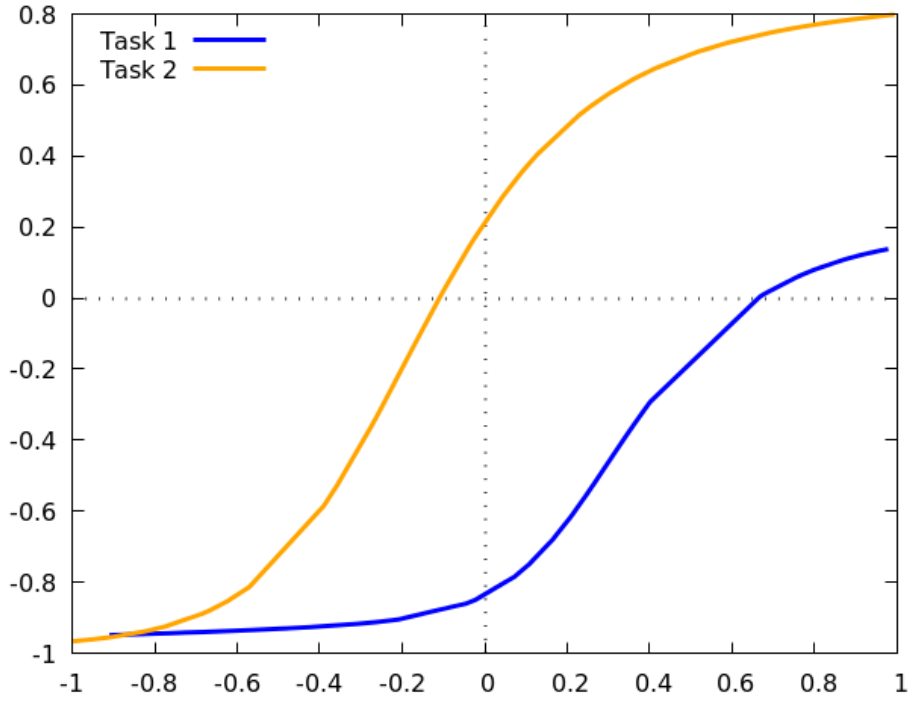

Figure 8.1 – Two alternating 2D functions to be learned, t_1 (blue) and t_2 (orange). The EA adapts a population of ANNs sequentially on each task during 30 generations per training period. This is repeated 4 times for both t_1 and t_2, for a total of 240 generations.

For each independent run of the experiment, our algorithm initializes its population with a set of random ANNs ($|\mathbf{P}| = 80$ in our case). They all share the same fixed structure, a sufficiently large random structure including 15 randomly added neurons and 30 randomly added connections. This is done to ensure that the structure possesses enough expressivity to properly approximate both functions. The algorithm adapts the ANNs in the population to the tasks, using the Euclidian error between (a) the current goal function and (b) the function approximated by each ANN, as fitness function to be minimized. At each generation, a new population of 80 offspring must be generated. With a probability $p = 0.5$, for each offspring, a parent is selected with a proportionate selection operator based on the average fitness of the lineages in the history, and it is kept for the next generation without mutating it. With a probability $1 - p = 0.5$, a parent is selected using fitness-proportionate selection based on the fitness function on the current task. The parent is then mutated using a Gaussian random variable with step-size $\sigma = 0.2$ to all the weights in the ANN. Every 30 generations of the algorithm, the function to be approximated changes to the other one, alternating between both tasks, and starting by the t_1 function (blue line in Figure 8.1). Each function is presented to the algorithm in 4 separate training periods, which makes a total of 2 functions $\times$ 4 training periods per function $\times$ 30 generations per training period $=$ 240 generations for each run of the experiment. The parameters of the experiments are shown in Table 8.1. We perform 10 independent runs.

Experiments	
Number of runs	10
p	0.5
Evolution length	240 generations
Frequency of task switch	Every 30 generations
Genome size	32 weights
Mutation step-size	$\sigma = 0.2$
Population size	80

Table 8.1 – Experimental settings for the experiments on avoiding forgetting.

Measures

We are interested in evaluating the quality of the approximation of both functions for the evolved solutions through each training period. We also focus on the efficiency of the the adaptation and the performance loss of the best solution in the population when the algorithm readapts to a previously seen function. In our experiments, readaptation refers to training on a task seen in the past. Each task is presented 4 times to the algorithm: a first pretraining phase, and 3 readaptation phases. Prior to each readaptation, training on the other function may have erased high-performing solutions in the population, which is what our algorithm tries to reduce. Consequently, for each independent run we take the following measures on the evolutionary process

- First, at each generation, we measure the best fitness in the population for each task. This provides information on the effectiveness (quality) and efficiency (speed) of the first training and following readaptations to each task.

- Second, we measure the speed of readaptation to each previously seen task, as the number of generations to reach a given target performance level with the best fitness in the population for each retraining period. In our experiments, we define the target level as an error of 0.05 for each training period. This allows us to evaluate the efficiency of readaptations, *i.e.* how fast the population readapts to a task that has been learned in the past.

8.3.1 Results

Figure 8.2 shows the fitness (regression error) of the best individual in the population for each task across generations. First, the random initial population is trained on Task 1, which accidentally slightly improves the performance on Task 2. Then, two oscillating patterns of the best fitness on each task can be seen through the alternating training periods on each task. When the population is first trained on t_2 (generations 30 to 60), the corresponding approximation error decreases, while the error on t_1 increases. This is also seen in the rest of training periods: the error on t_1 decreases while the error on t_2 increases, and vice versa (*e.g.* when learning t_1 for the second time (generations 60 to 90). This means that our algorithm does not completely eliminate forgetting in sequences of tasks.

A closer inspection of the results shows that forgetting is at least slightly reduced. Indeed, at generation 180 and generation 240, *i.e.* after the last two training periods on t_2, the best individual in the population has better fitness values in t_1 (~ 0.20 and ~ 0.15) than at generation 60, after the first training period on t_2 (~ 0.30). There is a similar, though smaller trend for t_2. The best fitness at the last training period on t_2 (generation 210) starts from ~ 0.25, while the

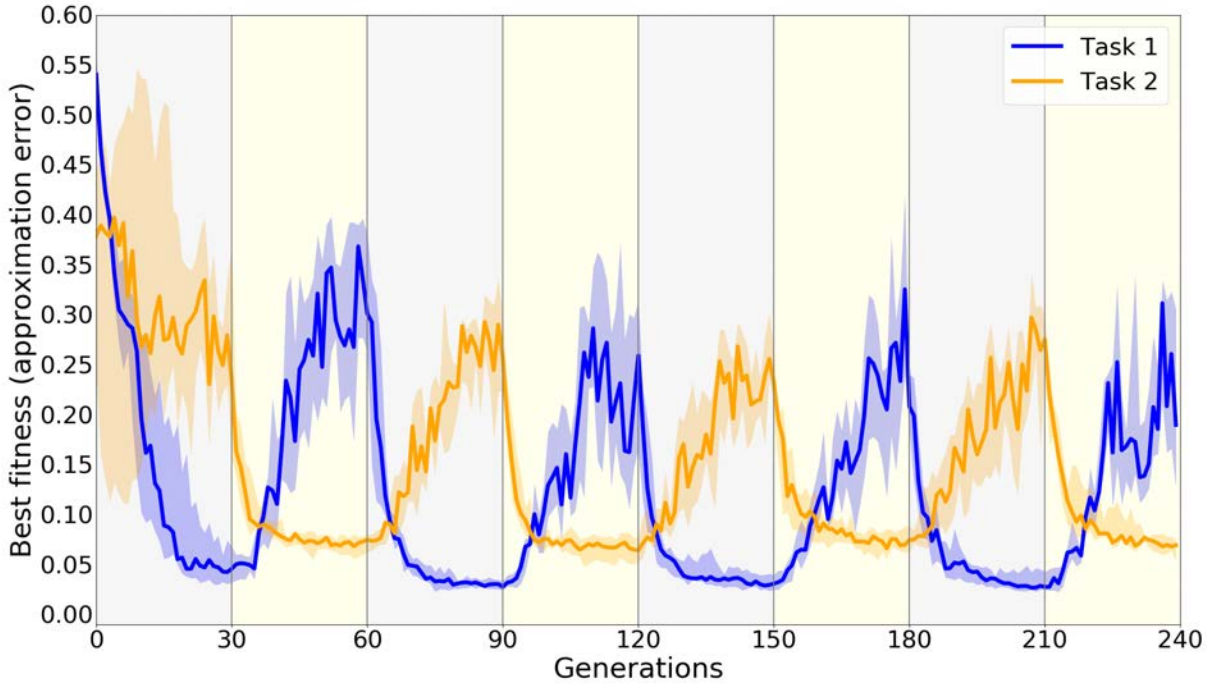

Figure 8.2 – Best fitness in the population (regression error) for each alternating task over generations in the experiments on incremental adaptation. The blue line (respectively orange line) corresponds to the median value of the 10 independent runs of the lowest approximation error in the population for the first task, t_1 (respectively t_2). Shaded areas represent the lower and upper quartiles of the best fitness on each run on the 10 independent runs. Vertical slices represent training periods on each task, switching every 30 generations.

first and second periods start from ~ 0.30. This shows that our algorithm limits forgetting to a certain level by retaining well performing individuals.

Additionally, readaptation, in terms of the best fitness in the population, seems faster for the two last learning periods on t_1 (generations 120 to 150, and 180 to 210) than for the two first periods (generations 0 to 30, and 60 to 90). However, this is not the case for t_2, for which readaptation seems slower during the latter training periods. To better evaluate the readaptation speed on each task, we measure the number of generations for the best individual to reach a target level of 0.05 at each training period (the first training and the three readaptation periods, *cf.* previous section). The results, which are different for t_1 and t_2, are shown in Figure 8.3. For t_1 (left), the time to reach the target level decreases with successive readaptation periods when compared to the pretraining, which indicate faster adaptivity when the task has been faced in the past. However, this is not the case for t_2 (right), in which such speed of readaptation seems to be similar in all training periods. This confirms our previous inspection on the best fitness over generations (Figure 8.2).

Our results show that there is an improvement on the speed of readaptation for t_1 over training periods, while, for t_2, forgetting is less avoided. Since the algorithm faces t_1 first, this could indicate a bias favoring the first task presented in the sequence. Thus, a legitimate question that needs further investigation is: is the order on which tasks are presented important with respect to how forgetting is avoided?

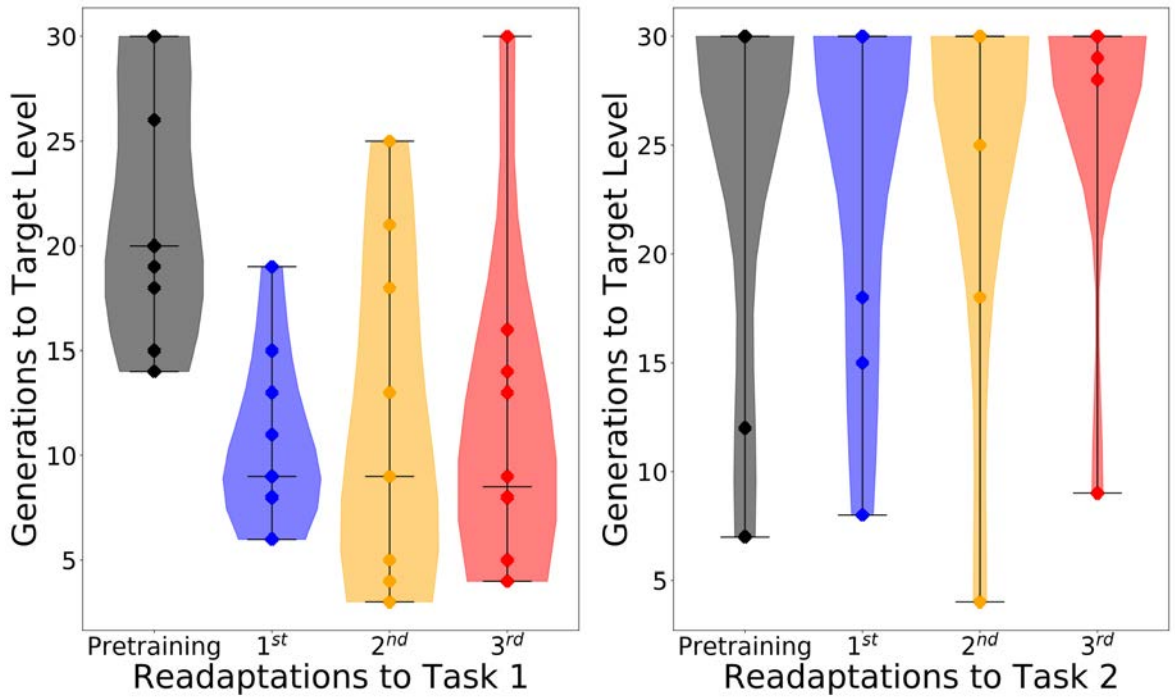

Figure 8.3 – Speed of readaptation as the number of generations to reach the target level of 0.05 with the best fitness of the population, for the four training periods of each task, *i.e.* the pretraining and the three readaptation periods (left: t_1, right: t_2). Violin plots in both graphs represent the kernel density function of the distribution over the 10 runs, and datapoints are added for reference.

Discussion

These results show that our approach can reduce forgetting by encouraging a diverse population that maintains individuals whose ancestors performed well in the past. That said, these results are preliminary, and more thorough investigation of the influence of different parameters should be done. For example, p, the probability of selecting genomes in the population with respect to their respective phylogenetic lineages, should have a strong influence on the ability of the algorithm to avoid forgetting. The frequency at which tasks in the sequence are presented to the adapting system is another important parameter, which could strongly influence the results of the algorithm. Rapidly switching tasks might not provide with enough time for the population to adequately adapt to the current task, while, if switching too slowly, individuals that performed well in the past could be completely erased. In the proposed algorithm, the objective function aggregating information from the past is the average fitness function of the ancestors in the lineage. More informed aggregation functions could be defined, which may help maintaining high-performing individuals from the past. Further, besides using fitness values of the ancestors, aggregation functions could also use measures of the difference between current individuals and their respective ancestors, in order to maintain in the population those that are similar to past high-performing individuals.

Additionally, instead of two alternating tasks, considering larger sequences could reveal further insights on the long-term populational adaptation to changing conditions. For instance, if a given task has not been faced for a long time, it could be gradually forgotten, unless specific rehearsal

mechanisms are in place to retrain on the task. Furthermore, the population in our algorithm has a limited size that could become saturated if a large number of different tasks are addressed, in which case forgetting of some of the tasks would become unavoidable. As such, either providing a sufficiently large population or progressively augmenting the size of the population could be possible approaches to allow for long-term adaptation when large sequences of tasks are addressed.

8.4 Toward Forgetting Avoidance in Distributed EER

Although the algorithm that we test in our experiments is centralized, it could be modified for the distributed settings of EER to address swarm robot adaptation to changing conditions. This is so because the required data for our algorithm to retain individuals that performed well in the past does not depend on global knowledge. In distributed EER, a robot broadcasts its active genome with the corresponding fitness value when meeting other robots. The lineage history information for each genome, *e.g.* the list of fitness values of the ancestors, can be kepts next to the genome in the local population of each robot. Consequently, the lineage of the active genome could be sent to nearby robots, who would store it in their respective local populations, and update it when generating new offspring.

These modifications to the algorithm make it possible to translate our approach to distributed EER, which could potentially allow for enhanced adaptivity in swarms of robotic agents when facing changing environments. This requires further investigation to ascertain if a distributed version of our algorithm provides similar results to those presented in the previous section.

8.5 Conclusion

In this chapter, we have presented a novel approach to the problem of retaining previous skills when adapting to changing conditions using EAs. Our approach focuses on allowing some candidate solutions in the population to survive because their ancestors performed well in the past, even if they are no longer competitive. Our proposed algorithm, FAEA, maintains lineage information, *e.g.* for each genome in the population, the fitness values of its ancestors. Selection is then probabilistically applied, based on either the fitness of the current task or on the performance of the ancestors in the lineages of the genomes in the population. As such, the algorithm maintains a diverse population that includes high-performing genomes in the current task, as well as genomes with high-performing lineages. When selecting with respect to such lineages, mutation is not applied, thus minimally disturbing these individuals over generations, which allows for the population to readapt faster.

We test our algorithm in a set of preliminary experiments, where a population of ANNs needs to adapt to an alternating sequence of two regression tasks, each consisting in approximating a 2D function. These preliminary results show the feasibility of our approach, *i.e.* addressing catastrophic forgetting at the level of populations in an EA. Our results show that forgetting can be reduced at the populational level by applying selection pressure toward individuals whose ancestors performed well. Further, retaining such individuals in the population allows for a slightly improved readaptation to previously learned tasks, when compared to learning from scratch. Our proposed algorithm represents progress toward evolutionary systems that avoid forgetting by maintaining a diverse population with individuals that performed well in the past. Although the results presented in this chapter are preliminary, they show that our proposed approach has potential capabilities for the incremental adaptation to changing conditions.

Additionally, we discuss how this algorithm could be modified to be applied in distributed EER settings, where a swarm of robots adapts online to changing conditions, *i.e.* environments or tasks. This is relatively straightforward, since information on the phylogenetic lineages could be stored along with the respective genomes in the local population of each robot, and this could be also broadcast at the same time as the genomes when robots meet.

Avoiding forgetting is an important topic when designing Lifelong Machine Learning systems that progressively adapt to their environment in an open-ended manner. Our approach provides a first step toward such adaptation to changing conditions in swarms of robots using distributed EER. That said, the experiments presented in this chapter are preliminary, and consider a sequence of abstract regression tasks in centralized settings. More thorough evaluation of the influence of the parameters in the algorithm (*e.g.* probability of selection on history, size of the population, frequency of task switching) and evaluation in swarm robotic tasks (*e.g.* individual or collaborative item collection) are required. This could help us to better understand the possibilities open by our approach related to the incremental evolutionary adaptation to changing conditions in swarms of robotic agents.

Finale

9

Conclusion and Perspectives

Contents

In this chapter, we conclude the thesis. We start by presenting a summary of the manuscript. We then list the contributions of our work on the adaptation of swarm robot behavior using distributed EER approaches. Subsequently, we discuss possible future perspectives in the light of the contributions of this thesis, which could improve adaptivity in swarms of robots when facing unknown environments. Finally, we provide some concluding remarks.

9.1 Summary

In this thesis, we have investigated online adaptation of autonomous behaviors in swarms of robotic agents to unknown environments using distributed Embodied Evolution approaches. These approaches belong to the field of Evolutionary Robotics, an algorithmic family that uses Evolutionary Algorithms to learn robot behaviors, by taking inspiration from natural evolution. The work in this thesis on online swarm robot adaptation belongs to Collective Adaptive Systems, and is at the crossroads of Swarm Intelligence and Machine Learning. We have presented the context of this thesis, while describing the notion of autonomous robots, the reactive agent perspective taken in our work, and the particularities of collective robot systems. We have described the problem of adaptation, which is the central focus of our work, and we have reviewed Machine Learning approaches with a focus on Reinforcement Learning for robotics, while describing different approaches in the literature that address sets of tasks over time. Subsequently, we have reviewed Evolutionary Robotics, *i.e.* applying evolutionary approaches to learn robot behavior, while developing our discussion on evolving swarm robot behavior. We have further provided a classification and survey of existing work using Embodied Evolutionary Robotics to evolve swarm behavior. We have then presented our approach for the online adaptation of swarm robot behavior, while stating the research questions in the thesis. Finally, we have described the contributions in our work as: (a) investigating the influence of selection pressure in distributed EER; (b) evolving collaborative behaviors in distributed EER; (c) augmenting the topology of neural controllers in

distributed EER; and (d) avoiding task forgetting at the populational level in neuroevolution. These are further detailed in the next section.

9.2 Contributions

The main contributions presented in this thesis aim at better understanding and enhancing adaptation mechanisms in robot swarms. Our goal is to promote adaptivity in such robot swarms when facing unknown and possibly changing environments. The approaches in this thesis fall into the family of distributed embodied evolutionary robotics, in which each robot in the swarm adapts to its environment using an evolutionary algorithm, while exchanging information about its learning when meeting other robots. Here, we summarize the contributions reported in this document, while highlighting the findings in our work.

Reconsidering Selection Pressure in Distributed EER

Traditionally, EAs increase the diversity in the population by adequately tuning selection pressure to better search for high performing solutions: selection pressure is often slightly lowered to enhance diversity and avoid premature convergence. However, in distributed EER approaches, the dynamics of the optimization algorithm are different than in classical centralized EAs. In such a distributed case, we have tested different levels of increasing selection pressure applied at the robot level, and evaluated the impact on the performance obtained by the algorithm. We have shown that, in distributed EER, applying extremely high levels of selection pressure improves learning: the higher the selection pressure, the better the performances. The reason stems from selection being applied on local subpopulations, which are built based on robot interactions. Since each individual robot does not interact with all the other robots in the swarm, local subpopulations are different. Therefore, diversity, which is needed for evolution to progress, is naturally maintained by these distributed subpopulations.

Distributed Evolution of Collaboration

We have investigated the distributed Embodied Evolution of behaviors for an intrinsically collaborative task, in which at least two robots are required to collaborate to solve the task. Learning collaborative behavior from scratch with decentralized approaches is particularly difficult, since coordination between robots needs to emerge. In these experiments, we have further complexified the task by considering different subtasks, which need different actions to be solved. This is done to emphasize the need for coordination between robots to solve the task. In our case, the challenge arises from the fact that an individual that tries to collaborate cannot benefit from such collaboration unless other individuals are already showing a collaborative behavior. Our results show that a swarm of robots using a distributed EE approach can adapt to solve such a collaborative task. Additionally, we show that, without any specific mechanism to enforce task distribution over the robots, the algorithm allows to learn collaborative behaviors that do not neglect any type of subtask. This opens perspectives of more complex tasks that could be addressed using distributed EER, such as tasks that need some emergence of global patterns among all the robots in the swarm.

Distributed Evolution of Neural Topologies

To increase the expressivity of neurocontrollers in swarms of robots, we consider evolving their structure. We have used a topology-evolving distributed EER that includes topological mutations,

such as adding neurons and connections. The algorithm needs to mark and keep track of the order of these new structures in the controller, which has been typically done in centralized algorithms by using global information, which is not available in distributed settings. In this sense, we have proposed Gene Clocks, a decentralized mechanism that takes inspiration from Distributed Computer Systems, and avoids resorting to global information. In our experiments, a swarm of robots progressively evolves the topology and the weights of their neurocontrollers. We have compared two marking mechanisms, one that uses global information, which should not be available to robots, and our proposed approach, which exclusively relies on local information. Our results show that our proposed method does not lead to a loss of performance, nor a quantitative difference in terms of size and diversity of the evolved controllers, while providing a truly decentralized mechanism for neural topology evolution that depends only on local information.

Avoiding Forgetting at the Populational Level

When agents adapt to a dynamic environment, previously acquired knowledge can be reused to readapt more efficiently to changes. To do so, such previous knowledge needs to be retained, which, for reactive agents with ANNs, may be challenging (*cf.* catastrophic forgetting, Chapter 2). To help avoid forgetting, we have proposed and tested an algorithm to incrementally adapt populations of neural networks to changing tasks that limits forgetting at the level of populations, thus promoting and improving adaptivity. This approach is based on maintaining a trace of historical information (such as fitness values, genomes, or behavioral descriptors) of the ancestors of the individuals in the population, using a phylogenetic tree structure. The lineages of each genome in the current population are then exploited by the EA during reproduction to slightly bias selection and variation toward individuals whose ancestors performed well, *e.g.* in a previous task. In a set of preliminary experiments, we have tested the approach in a simple problem consisting on approximating two alternating 2D functions, using a centralized EA. Our results show the feasibility of our algorithm, which provides a promising novel approach to limit catastrophic forgetting at the level of populations of agents that adapt to changing environments.

9.3 Perspectives

Adaptation in swarms of robots is an important topic in current research that has numerous potential future applications and could have a large impact in society. These applications include, for example, oceanic and planetary exploration and monitoring, where the possibly changing conditions cannot be foreseen and human supervision is not possible. Another example of potential application concerns robot rescue in disaster areas, which may be inaccessible to human rescue teams and requires adaptive robots to succeed. In addition, deployment of mobile communication networks, where each robot in the swarm acts as a mobile relay for information, would benefit from adaptation mechanisms to allow for reconfiguration in the event of changing conditions in the environment. Many questions remain open and, while investigating adaptation in our work, we have identified several potential perspectives, in both the short and the long term. In the short-term, some of our experiments have shown that there is room for improvement in current EER approaches. Long-term perspectives aim at further enhancing swarm robot adaptivity to provide experimenters with a set of algorithmic tools that could be used when designing such systems. Here, we list and discuss both short-term and long-term perspectives in the light of the findings of this thesis.

Considering Robots and Controllers to be Adapted

Mechanisms for behavior adaptation shift the role of human experimenters in the design of swarms of robots, from designing swarm behavior directly to designing the process of adaptation that shapes this behavior. While distributed EER approaches allow for behavior adaptation in swarms of robots, attention should be paid to the possibilities left for adaptation to tune. Indeed, if excessive possibilities are provided to the adaptation process, the search space becomes larger, thus possibly hindering adaptation. On the other hand, if too narrow possibilities are left for adaptation to tune, appropriate behaviors for the task at hand may be unattainable, or at least this could bias the adaptation process toward a limited set of task-solving behaviors. For example, in our work, the architecture of the controller, or at least part of it, *e.g.* the set of sensors and actuators, have been specified *a priori*.

As an example, in Chapter 6, while investigating the distributed evolution of collaboration, we have identified some inefficiencies in the behaviors. To perform the collaborative task, pairs of robots need to simultaneously select a particular action when jointly reaching items of different types in the environment. Upon analysis of the results of the experiment, we have discovered that the design in the neurocontroller of the effector output used to choose such action is actually biased, thus making some subtasks (collecting some types of items) easier than others. This is a result of the design of the effector output in the controller: a different encoding of such output in the controller could probably lead to more balanced difficulty over the subtasks.

Furthermore, the design of the environment and the robot morphology can also bias or constrain the possible behaviors to evolve. For example, in the same experiments, if we increase the frequency at which the tasks become available for difficult or rarer subtasks, would the distribution between subtasks be more balanced? Density, spatial dispersion, and proportion per type of subtask could be varied to analyze the impact of environmental pressures on the evolved strategies. Additionally, when providing reward feedback to the robots in the same experiments, a lower reward is given when collaborating in larger groups, which has led to behaviors where robots only collaborate in pairs. An different design of the reward feedback to evaluate behaviors could yield different evolved strategies.

To summarize, although adaptation mechanisms for robot swarms may be powerful, this does not relieve the human experimenter from designing appropriate elements to make such adaptation possible. This includes robot and controller architectures, reward functions, and environmental shaping mechanisms.

More Informed Topological Alignment

In Chapter 7, we have used odNEAT to adapt the structure and weights of neurocontrollers in robot swarms. odNEAT includes the crossover operator used by NEAT, which aligns two genomes based on their historical innovation numbers, and classifies genes (neurons and synaptic connections) into three categories: matching, excess and disjoint. Such a classification omits a more detailed information included in the genome: innovation numbers contain information on the chronological relative age of genes in the genomes.

We are interested in investigating if it is possible to further exploit this information to define more informed crossover operators or genotypic distance measures. On the one hand, these could allow for better mixing functional building blocks in the evolving controllers, taking advantage from fit modules of both parent networks. On the other hand, the niching mechanism, which is based on the computation of a genotypic distance, could be more precise, and include in the same species controllers that encode similar behaviors, thus protecting innovations at a functional level.

Adaptive Operators in Distributed EER

In the experiments reported in this thesis, evolutionary operators, *e.g.* selection and mutation, remain fixed through the process. Modulating the intensity of such operators could improve the adaptation of controllers, by making it either more efficient or more effective. For example, if a swarm adapts to a changing environment, when changes occur, it could be beneficial for mutation to explore further or for the intensity of selection pressure to increase, to allow for adequate adaptivity. Conversely, if a robot swarm has already appropriately adapted to a given environment, there is no such a need for exploration. Excessively strong mutation operators in this case could hinder the process, temporarily moving search away from behaviors that are already fit.

Avoiding Forgetting in Swarms of Robots

The algorithm presented in Chapter 7 allows for truly decentralized evolution of neural networks with augmenting topologies using Embodied Evolutionary Robotics. One possibility that topology evolution opens is to increase the storing capacity of individual robot neurocontrollers in the distributed population. As such, augmenting the controller structure might allow each individual robot to store information related to several conditions. A robot swarm could use such method to adapt to changing environments or tasks, while retaining information from previous conditions in the controller of each robot, *i.e.* avoiding forgetting at the individual level. This, in turn, could allow for an improved adaptivity due to such a lifelong learning, where the robots could efficiently readapt to conditions seen in the past.

In Chapter 8, we have proposed an algorithm that avoids forgetting at a populational level, and allows for readaptation to changing conditions by maintaining fit individuals from the past in the evolving population. The algorithm has been validated in a centralized offline settings for two tasks consisting in approximating 2D functions. To further validate this approach, we are interested in adapting the algorithm to distributed EER, and evaluating it on a swarm of robots sequentially facing changing environments and tasks. This could allow to avoid forgetting at the populational level across a swarm of robots that accumulate skills in incremental manner when facing such changing conditions.

Further, avoiding forgetting at both the individual and the populational level is not incompatible. We would be interested in merging both approaches, individual retention and populational retention, in distributed EE, to further improve the adaptivity of the swarm.

Complex Tasks

Item collection and navigation can be considered as relatively simple tasks. We believe that more complex and challenging tasks could provide further insights on the evolutionary dynamics in the distributed EER case, *e.g.* related to the impact of selection pressure, or the structural controller evolution. Particularly, tasks involving deceptive fitness functions, *e.g.* requiring specific actions to be performed in a particular order, could be of special interest. In such cases, the evolutionary search requires powerful and possibly adaptive operators to escape plateaus, saddle points and local valleys in the fitness landscape, which are typical of real-world problems. Further, applying auxiliary selection pressures in distributed EER, as done in mainstream ER, could enhance the search, and make evolution able to adapt swarm behaviors to unknown complex and possibly changing environments or tasks.

Another class of complex problems that could be addressed with distributed EER concerns tasks that require globally emergent behaviors. Learning of such collective emergent tasks, where

the task-solving behavior emerges at a global level from local interactions of the individual robots in the swarm, is already a difficult challenge in centralized offline settings. Distributed settings, including partial populations, online adaptation and local communication, make the evolution of such emergent behaviors even more challenging.

In this thesis, we have investigated evolutionary adaptation in swarms of robots using distributed approaches. These approaches fall in the field of distributed Embodied Evolutionary Robotics, and aim at adapting behaviors online for swarms of robots when facing unknown and possibly changing environments. While the scope of application and the possibilities of these approaches are large, they pose additional algorithmic and physical constraints to the robots in the system. This requires tailored algorithmic tools that are well adapted to the specific properties of distributed EER. The contributions in this thesis have improved the understanding of such distributed EER mechanisms for adapting swarms of robots. These findings have allowed us to identify further research questions to improve the adaptivity of swarms of robots. This demonstrates that Embodied Evolutionary Robotics is a powerful approach for the automatic adaptation of behaviors in swarms of robotic agents.

List of Publications

The contributions presented in this thesis have led to several scientific publications in both international and national (French) peer-reviewed conferences. Here, we present the list of publications for quick reference:

International peer-reviewed conferences:

ALIFE'14: Fernández Pérez, I., Boumaza, A., and Charpillet, F. (2014). Comparison of selection methods in on-line distributed evolutionary robotics. In *Proceedings of the International Conference on the Synthesis and Simulation of Living Systems (Alife'14)*, pages 282–289, New York. MIT Press

GECCO 2015: Fernández Pérez, I., Boumaza, A., and Charpillet, F. (2015a). Decentralized innovation marking for neural controllers in embodied evolution. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, GECCO '15, pages 161–168, Madrid, Spain. ACM

ECAL 2017: Fernández Pérez, I., Boumaza, A., and Charpillet, F. (2017). Learning Collaborative Foraging in a Swarm of Robots using Embodied Evolution. In *ECAL 2017 – 14th European Conference on Artificial Life*, Lyon, France. Inria. Nominated to the best paper award at ECAL2017 (4 nominees over 100+ papers)

National peer-reviewed conferences:

RJCIA 2015: Fernández Pérez, I., Boumaza, A., and Charpillet, F. (2015b). Influence of selection pressure in online, distributed evolutionary robotics. In *Actes des Rencontres nationales des Jeunes Chercheurs en Intelligence Artificielle (RJCIA 2015)*, pages 31–36, Rennes, France

Bibliography

[Alba and Tomassini, 2002] Alba, E. and Tomassini, M. (2002). Parallelism and evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 6(5):443–462.

[Alpaydin, 2010] Alpaydin, E. (2010). *Introduction to Machine Learning*. The MIT Press, 2nd edition.

[Amato *et al.*, 2016] Amato, C., Konidaris, G., Anders, A., Cruz, G., How, J. P., and Kaelbling, L. P. (2016). Policy search for multi-robot coordination under uncertainty. *The International Journal of Robotics Research*, 35(14):1760–1778.

[Amato *et al.*, 2015] Amato, C., Konidaris, G., Omidshafiei, S., Agha-mohammadi, A.-a., How, J. P., and Kaelbling, L. P. (2015). Probabilistic planning for decentralized multi-robot systems. In *2015 AAAI Fall Symposium Series*.

[Ammar *et al.*, 2014] Ammar, H. B., Eaton, E., Ruvolo, P., and Taylor, M. (2014). Online multi-task learning for policy gradient methods. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1206–1214.

[Ampatzis *et al.*, 2006] Ampatzis, C., Tuci, E., Trianni, V., and Dorigo, M. (2006). Evolution of signalling in a group of robots controlled by dynamic neural networks. In *International Workshop on Swarm Robotics*, pages 173–188. Springer.

[Anderson *et al.*, 2013] Anderson, S., Bredèche, N., Eiben, A., Kampis, G., and van Steen, M. (2013). Adaptive collective systems: herding black sheep. *BookSprints for ICT Research*.

[Angeline *et al.*, 1994] Angeline, P. J., Saunders, G. M., and Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *Neural Networks, IEEE Transactions on*, 5(1):54–65.

[Argyriou *et al.*, 2007] Argyriou, A., Evgeniou, T., and Pontil, M. (2007). Multi-task feature learning. In *Advances in neural information processing systems*, pages 41–48.

[Auerbach and Bongard, 2011] Auerbach, J. and Bongard, J. C. (2011). Evolving monolithic robot controllers through incremental shaping. *New Horizons in Evolutionary Robotics*, 341:55–65.

[Back, 1994] Back, T. (1994). Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, pages 57–62.

[Balakrishnan and Honavar, 1995] Balakrishnan, K. and Honavar, V. (1995). Properties of genetic representations of neural architectures. In *Proceedings of the World Congress on Neural Networks, WCNN '95*.

[Baldassarre *et al.*, 2003] Baldassarre, G., Nolfi, S., and Parisi, D. (2003). Evolving mobile robots able to display collective behaviors. *Artificial life*, 9(3):255–267.

[Baldassarre *et al.*, 2004] Baldassarre, G., Parisi, D., and Nolfi, S. (2004). Coordination and behaviour integration in cooperating simulated robots. In *From Animals to Animats V III. Proceedings of the 8 th International Conference on Simulation of Adaptive Behavior*. Citeseer.

[Bangel and Haasdijk, 2017] Bangel, S. and Haasdijk, E. (2017). Reweighting rewards in embodied evolution to achieve a balanced distribution of labour. In *ECAL 2017 – 14th European Conference on Artificial Life*, Lyon, France. Inria.

[Baray, 1997] Baray, C. (1997). Evolving cooperation via communication in homogeneous multiagent systems. In *Proceedings of Intelligent Information Systems, IIS'97*, pages 204–208. IEEE.

[Bayindir and Şahin, 2007] Bayindir, L. and Şahin, E. (2007). A review of studies in swarm robotics. *Turkish Journal of Electrical Engineering & Computer Sciences*, 15(2):115–147.

[Beer and Gallagher, 1992] Beer, R. D. and Gallagher, J. C. (1992). Evolving dynamical neural networks for adaptive behavior. *Adaptive behavior*, 1(1):91–122.

[Berger-Tal and Avgar, 2012] Berger-Tal, O. and Avgar, T. (2012). The glass is half-full: overestimating the quality of a novel environment is advantageous. *PLoS One*, 7(4).

[Bernard *et al.*, 2015] Bernard, A., André, J.-B., and Bredèche, N. (2015). Evolution of cooperation in evolutionary robotics : the tradeoff between evolvability and efficiency. In *Proceedings of the European Conference on Artificial Life 2015*, pages 495–502.

[Bernard *et al.*, 2016] Bernard, A., André, J.-B., and Bredèche, N. (2016). To cooperate or not to cooperate: Why behavioural mechanisms matter. *PLOS Computational Biology*, 12(5):1–14.

[Bonabeau *et al.*, 1999] Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, Inc., New York, NY, USA.

[Bonabeau *et al.*, 1996] Bonabeau, E., Théraulaz, G., and Deneubourg, J.-L. (1996). Quantitative study of the fixed threshold model for the regulation of division of labour in insect societies. *Proceedings of the Royal Society of London B: Biological Sciences*, 263(1376):1565–1569.

[Bongard and Lipson, 2004] Bongard, J. and Lipson, H. (2004). Once more unto the breach: Co-evolving a robot and its simulator. In *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9)*, pages 57–62.

[Bongard, 2000] Bongard, J. C. (2000). Reducing collective behavioural complexity through heterogeneity. In *Artificial Life VII: Proceedings of the Seventh International Conference*, pages 327–336.

[Brambilla *et al.*, 2013] Brambilla, M., Ferrante, E., Birattari, M., and Dorigo, M. (2013). Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41.

[Branke, 1999] Branke, J. (1999). Memory-enhanced evolutionary algorithms for changing optimization problems. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99)*, volume 3, pages 1875–1882. IEEE.

[Branke *et al.*, 2000] Branke, J., Kaußler, T., Smidt, C., and Schmeck, H. (2000). A multi-population approach to dynamic optimization problems. In *Evolutionary Design and Manufacture*, pages 299–307. Springer.

[Bredèche, 2014] Bredèche, N. (2014). Embodied evolutionary robotics with large number of robots. In *Proceedings of the 14th international conference on the synthesis and simulation of living systems (ALIFE 14)*, pages 1–2.

[Bredèche *et al.*, 2010] Bredèche, N., Haasdijk, E., and Eiben, A. (2010). On-line, on-board evolution of robot controllers. In *Artifical Evolution*, pages 110–121. Springer.

[Bredèche *et al.*, 2015] Bredèche, N., Haasdijk, E., and Prieto, A. (2015). Elements of embodied evolutionary robotics. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO Companion '15, pages 1247–1247, New York, NY, USA. ACM.

[Bredèche and Montanier, 2010] Bredèche, N. and Montanier, J.-M. (2010). Environment-driven Embodied Evolution in a Population of Autonomous Agents. In *Parallel Problem Solving from Nature, PPSN 2010*, pages 290–299, Krakow, Poland.

[Bredèche *et al.*, 2017] Bredèche, N., Montanier, J.-M., and Carrignon, S. (2017). Benefits of proportionate selection in embodied evolution: A case study with behavioural specialization. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, pages 1683–1684, New York, NY, USA. ACM.

[Bredèche *et al.*, 2012] Bredèche, N., Montanier, J.-M., Liu, W., and Winfield, A. (2012). Environment-driven Distributed Evolutionary Adaptation in a Population of Autonomous Robotic Agents. *Mathematical and Computer Modelling of Dynamical Systems*, 18(1):101–129.

[Bredèche *et al.*, 2013] Bredèche, N., Montanier, J.-M., Weel, B., and Haasdijk, E. (2013). Roborobo! a fast robot simulator for swarm and collective robotics. *CoRR*, abs/1304.2888.

[Caruana, 1993] Caruana, R. (1993). Multitask learning: A knowledge-based source of inductive bias. In *Machine Learning, Proceedings of the Tenth International Conference, University of Massachusetts, Amherst, MA, USA*, pages 41–48.

[Caruana, 1994] Caruana, R. (1994). Multitask connectionist learning. In *Proceedings of the Connectionist Models Summer School*, page 372. Psychology Press.

[Caruana, 1997] Caruana, R. (1997). Multitask learning. *Machine Learning*, 28(1):41–75.

[Chapelle *et al.*, 2010] Chapelle, O., Schlkopf, B., and Zien, A. (2010). *Semi-Supervised Learning*. The MIT Press.

[Chatzilygeroudis *et al.*, 2017] Chatzilygeroudis, K. I., Rama, R., Kaushik, R., Goepp, D., Vassiliades, V., and Mouret, J. (2017). Black-box data-efficient policy search for robotics. *CoRR*, abs/1703.07261.

[Chen *et al.*, 2016] Chen, Z., Hruschka, Jr., E. R., and Liu, B. (2016). Lifelong machine learning and computer reading the web. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 2117–2118, New York, NY, USA. ACM.

[Chow, 2004] Chow, R. (2004). Evolving genotype to phenotype mappings with a multiple-chromosome genetic algorithm. In *Genetic and Evolutionary Computation Conference*, pages 1006–1017. Springer.

[Christensen and Dorigo, 2006] Christensen, A. L. and Dorigo, M. (2006). Evolving an integrated phototaxis and hole-avoidance behavior for a swarm-bot. In *Proceedings of the 10th International Conference on the Simulation and Synthesis of Living Systems (Alife X)*, pages 248–254.

[Clune *et al.*, 2009] Clune, J., Beckmann, B. E., Pennock, R. T., and Ofria, C. (2009). Hybrid: A hybridization of indirect and direct encodings for evolutionary computation. In *European Conference on Artificial Life*, pages 134–141. Springer.

[Clune *et al.*, 2013] Clune, J., Mouret, J.-B., and Lipson, H. (2013). The evolutionary origins of modularity. *Proceedings of the Royal Society B*, 280:20122863.

[Clune *et al.*, 2011] Clune, J., Stanley, K. O., Pennock, R. T., and Ofria, C. (2011). On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation*, 15(3):346–367.

[Cockburn and Kobti, 2009] Cockburn, D. and Kobti, Z. (2009). Agent specialization in complex social swarms. In *Innovations in Swarm Intelligence*, pages 77–89. Springer.

[Cohen *et al.*, 2007] Cohen, J. D., McClure, S. M., and Angela, J. Y. (2007). Should i stay or should i go? how the human brain manages the trade-off between exploitation and exploration. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 362(1481):933–942.

[Conte *et al.*, 2004] Conte, D., Foggia, P., Sansone, C., and Vento, M. (2004). Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, 18(03):265–298.

[Couceiro *et al.*, 2013] Couceiro, M. S., Rocha, R. P., and Ferreira, N. M. (2013). A PSO multi-robot exploration approach over unreliable manets. *Advanced Robotics*, 27(16):1221–1234.

[Coulouris *et al.*, 2011] Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. (2011). *Distributed Systems: Concepts and Design*. Addison-Wesley, 5th edition.

[Cozman *et al.*, 2003] Cozman, F. G., Cohen, I., and Cirelo, M. C. (2003). Semi-supervised learning of mixture models. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 99–106.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

[Darwin, 1859] Darwin, C. (1859). *On the Origin of the Species by Means of Natural Selection: Or, The Preservation of Favoured Races in the Struggle for Life*. John Murray.

[De Jong *et al.*, 2001] De Jong, E. D., Watson, R. A., and Pollack, J. B. (2001). Reducing bloat and promoting diversity using multi-objective methods. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 11–18. Morgan Kaufmann Publishers Inc.

[De Jong, 1975] De Jong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, Ann Arbor, MI, USA.

[Deisenroth and Rasmussen, 2011] Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472.

[Deisenroth *et al.*, 2015] Deisenroth, M. P., Fox, D., and Rasmussen, C. E. (2015). Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423.

[Deisenroth *et al.*, 2013] Deisenroth, M. P., Neumann, G., and Peters, J. (2013). A survey on policy search for robotics. *Found. Trends Robot*, 2:1–142.

[Deneubourg and Goss, 1989] Deneubourg, J.-L. and Goss, S. (1989). Collective patterns and decision-making. *Ethology Ecology & Evolution*, 1(4):295–311.

[Di Mario *et al.*, 2013] Di Mario, E., Talebpour, Z., and Martinoli, A. (2013). A comparison of PSO and reinforcement learning for multi-robot obstacle avoidance. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 149–156. Ieee.

[Dibangoye *et al.*, 2009] Dibangoye, J. S., Mouaddib, A.-I., and Chai-draa, B. (2009). Point-based incremental pruning heuristic for solving finite-horizon Dec-POMDPs. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 569–576. International Foundation for Autonomous Agents and Multiagent Systems.

[Dinu *et al.*, 2013] Dinu, C. M., Dimitrov, P., Weel, B., and Eiben, A. (2013). Self-adapting fitness evaluation times for on-line evolution of simulated robots. In *Proc. of GECCO'13*, pages 191–198. ACM.

[Doncieux *et al.*, 2015] Doncieux, S., Bredèche, N., Mouret, J.-B., and Eiben, A. E. G. (2015). Evolutionary robotics: what, why, and where to. *Frontiers in Robotics and AI*, 2:4.

[Doncieux and Mouret, 2010] Doncieux, S. and Mouret, J.-B. (2010). Behavioral diversity measures for Evolutionary Robotics. In *Congress on Evolutionary Computation (CEC), at the WCCI 2010 IEEE World Congress on Computational Intelligence*, pages 1303–1310, Espagne.

[Doncieux and Mouret, 2014] Doncieux, S. and Mouret, J.-B. (2014). Beyond black-box optimization: a review of selective pressures for evolutionary robotics. *Evolutionary Intelligence*, 7(2):71–93.

[Dorigo *et al.*, 2006] Dorigo, M., Birattari, M., and Stutzle, T. (2006). Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39.

[Dorigo and Colombetti, 1994] Dorigo, M. and Colombetti, M. (1994). Robot shaping: developing autonomous agents through learning. *Artif. Intell.*, 71(2):321–370.

[Ducatelle *et al.*, 2009] Ducatelle, F., Förster, A., Di Caro, G. A., and Gambardella, L. M. (2009). New task allocation methods for robotic swarms. In *9th IEEE/RAS Conference on Autonomous Robot Systems and Competitions.*

[Eberbach, 2002] Eberbach, E. (2002). On expressiveness of evolutionary computation: Is EC algorithmic? In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 1, pages 564–569. IEEE.

[Eiben *et al.*, 2010a] Eiben, A., Haasdijk, E., and Bredèche, N. (2010a). Embodied, On-line, On-board Evolution for Autonomous Robotics. In P. Levi, S. K. E., editor, *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution.*, volume 7 of *Series: Cognitive Systems Monographs*, pages 361–382. Springer.

[Eiben *et al.*, 2010b] Eiben, A., Karafotias, G., and Haasdijk, E. (2010b). Self-adaptive mutation in on-line, on-board evolutionary robotics. In *Fourth IEEE Int. Conf. on Self-Adaptive and Self-Organizing Systems Workshop (SASOW)*, pages 147–152. IEEE.

[Eiben *et al.*, 2006] Eiben, A. E., Nitschke, G. S., and Schut, M. C. (2006). Collective specialization for evolutionary design of a multi-robot system. In *International Workshop on Swarm Robotics*, pages 189–205. Springer.

[Eiben and Schippers, 1998] Eiben, A. E. and Schippers, C. A. (1998). On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35(1-4):35–50.

[Eiben and Smith, 2003] Eiben, A. E. and Smith, J. E. (2003). *Introduction to Evolutionary Computing*. Springer.

[Eker *et al.*, 2011] Eker, B., Özkucur, E., Meriçli, C., Meriçli, T., and Akin, H. L. (2011). A finite horizon Dec-POMDP approach to multi-robot task learning. In *Application of Information and Communication Technologies (AICT), 2011 5th International Conference on*, pages 1–5. IEEE.

[Elfwing, 2007] Elfwing, S. (2007). *Embodied evolution of learning ability*. PhD thesis, KTH.

[Elfwing *et al.*, 2011] Elfwing, S., Uchibe, E., Doya, K., and Christensen, H. I. (2011). Darwinian embodied evolution of the learning ability for survival. *Adaptive Behavior*, 19(2):101–120.

[Ester *et al.*, 1996] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., *et al.* (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231.

[Evgeniou and Pontil, 2004] Evgeniou, T. and Pontil, M. (2004). Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 109–117. ACM.

[Fansi-Tchango *et al.*, 2014] Fansi-Tchango, A., Thomas, V., Buffet, O., Dutech, A., and Flacher, F. (2014). Tracking multiple interacting targets using a joint probabilistic data association filter. In *Information Fusion (FUSION), 2014 17th International Conference on*, pages 1–8. IEEE.

[Fernández Pérez *et al.*, 2014] Fernández Pérez, I., Boumaza, A., and Charpillet, F. (2014). Comparison of selection methods in on-line distributed evolutionary robotics. In *Proceedings of the International Conference on the Synthesis and Simulation of Living Systems (Alife'14)*, pages 282–289, New York. MIT Press.

[Fernández Pérez *et al.*, 2015a] Fernández Pérez, I., Boumaza, A., and Charpillet, F. (2015a). Decentralized innovation marking for neural controllers in embodied evolution. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, GECCO '15, pages 161–168, Madrid, Spain. ACM.

[Fernández Pérez *et al.*, 2015b] Fernández Pérez, I., Boumaza, A., and Charpillet, F. (2015b). Influence of selection pressure in online, distributed evolutionary robotics. In *Actes des Rencontres nationales des Jeunes Chercheurs en Intelligence Artificielle (RJCIA 2015)*, pages 31–36, Rennes, France.

[Fernández Pérez *et al.*, 2017] Fernández Pérez, I., Boumaza, A., and Charpillet, F. (2017). Learning Collaborative Foraging in a Swarm of Robots using Embodied Evolution. In *ECAL 2017 – 14th European Conference on Artificial Life*, Lyon, France. Inria. Nominated to the best paper award at ECAL2017 (4 nominees over 100+ papers).

[Ferrante *et al.*, 2015] Ferrante, E., Turgut, A. E., Duéñez-Guzmán, E., Dorigo, M., and Wenseleers, T. (2015). Evolution of self-organized task specialization in robot swarms. *PLoS computational biology*, 11(8).

[Ferrauto *et al.*, 2013] Ferrauto, T., Parisi, D., Di Stefano, G., and Baldassarre, G. (2013). Different genetic algorithms and the evolution of specialization: A study with groups of simulated neural robots. *Artificial life*, 19(2):221–253.

[Floreano *et al.*, 2008] Floreano, D., Dürr, P., and Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62.

[Floreano *et al.*, 2007] Floreano, D., Mitri, S., Magnenat, S., and Keller, L. (2007). Evolutionary conditions for the emergence of communication in robots. *Current biology*, 17(6):514–519.

[Fogel, 1995] Fogel, D. B. (1995). Phenotypes, genotypes, and operators in evolutionary computation. In *IEEE International Conference on Evolutionary Computation*, volume 1, page 193. IEEE.

[Francesca and Birattari, 2016] Francesca, G. and Birattari, M. (2016). Automatic design of robot swarms: Achievements and challenges. *Frontiers in Robotics and AI*, 3:29.

[French, 1992] French, R. M. (1992). Semi-distributed representations and catastrophic forgetting in connectionist networks. *Connection Science*, 4(3-4):365–377.

[French, 1999] French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128 – 135.

[From *et al.*, 2016] From, P. J., Gravdahl, J. T., and Pettersen, K. Y. (2016). *Vehicle-manipulator systems*. Springer.

[Gerkey and Matarić, 2004] Gerkey, B. P. and 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.

[Ghosh-Dastidar and Adeli, 2009] Ghosh-Dastidar, S. and Adeli, H. (2009). Spiking neural networks. *International journal of neural systems*, 19(04):295–308.

[Glover, 1989] Glover, F. (1989). Tabu search - part I. *ORSA Journal on computing*, 1(3):190–206.

[Goldberg and Deb, 1991] Goldberg, D. E. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann.

[Goldberg and Richardson, 1987] Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*, pages 41–49, Hillsdale, NJ, USA. L. Erlbaum Associates Inc.

[Gomes *et al.*, 2015] Gomes, J., Mariano, P., and Christensen, A. L. (2015). Novelty-driven cooperative coevolution. *Evolutionary computation*.

[Gomez and Miikkulainen, 1997] Gomez, F. and Miikkulainen, R. (1997). Incremental Evolution of Complex General Behavior. *Adaptive Behaviour*, 5:317–342.

[Greene *et al.*, 2006] Greene, K., Cooper, D. G., Buczak, A. L., Czajkowski, M., Vagle, J. L., and Hofmann, M. O. (2006). *Cognitive Agents for Sense and Respond Logistics*, pages 104–120. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Gruau, 1992] Gruau, F. (1992). Genetic synthesis of boolean neural networks with a cell rewriting developmental process. In *Combinations of Genetic Algorithms and Neural Networks, 1992., COGANN-92. International Workshop on*, pages 55–74. IEEE.

[Gruau, 1993] Gruau, F. (1993). Genetic synthesis of modular neural networks. In *GECCO'93*, pages 318–325. Morgan Kaufmann.

[Gu *et al.*, 2016] Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. (2016). Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838.

[Haasdijk and Bredèche, 2013] Haasdijk, E. and Bredèche, N. (2013). Controlling task distribution in MONEE. In Lió, P., Miglino, O., Nicosia, G., Nolfi, S., and Pavone, M., editors, *Advances In Artificial Life, ECAL 2013*, pages 671–678. MIT Press.

[Haasdijk *et al.*, 2014] Haasdijk, E., Bredèche, N., and Eiben, A. E. (2014). Combining environment-driven adaptation and task-driven optimisation in evolutionary robotics. *PLOS ONE*, 9(6):1–14.

[Haasdijk *et al.*, 2013] Haasdijk, E., Weel, B., and Eiben, A. E. (2013). Right on the monee: Combining task- and environment-driven evolution. In *Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference*, GECCO '13, pages 207–214, New York, NY, USA. ACM.

[Hamann *et al.*, 2011] Hamann, H., Schmickl, T., and Crailsheim, K. (2011). Coupled inverted pendulums: a benchmark for evolving decentral controllers in modular robotics. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 195–202. ACM.

[Hamann *et al.*, 2010] Hamann, H., Stradner, J., Schmickl, T., and Crailsheim, K. (2010). Artificial hormone reaction networks: Towards higher evolvability in evolutionary multi-modular robotics. *arXiv preprint arXiv:1011.3912*.

[Hansen *et al.*, 2010] Hansen, N., Auger, A., Finck, S., and Ros, R. (2010). Real-Parameter Black-Box Optimization Benchmarking 2010: Experimental Setup. Rapport de recherche, INRIA.

[Hansen and Ostermeier, 1996] Hansen, N. and Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. pages 312–317. Morgan Kaufmann.

[Hardin, 1968] Hardin, G. (1968). The tragedy of the commons. *Science*, 162(3859):1243–1248.

[Hastie *et al.*, 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). Overview of supervised learning. In *The elements of statistical learning*. Springer.

[Hauert *et al.*, 2014] Hauert, S., Mitri, S., Keller, L., and Floreano, D. (2014). Evolving cooperation: From biology to engineering. In *The horizons of evolutionary robotics*. MIT Press.

[Hauert *et al.*, 2009] Hauert, S., Zufferey, J.-C., and Floreano, D. (2009). Evolved swarming without positioning information: an application in aerial communication relay. *Autonomous Robots*, 26(1):21–32.

[Heinerman *et al.*, 2015a] Heinerman, J., Drupsteen, D., and Eiben, A. E. (2015a). Three-fold adaptivity in groups of robots: The effect of social learning. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 177–183. ACM.

[Heinerman *et al.*, 2015b] Heinerman, J., Rango, M., and Eiben, A. E. (2015b). Evolution, individual learning, and social learning in a swarm of real robots. In *Computational Intelligence, 2015 IEEE Symposium Series on*, pages 1055–1062.

[Hester *et al.*, 2010] Hester, T., Quinlan, M., and Stone, P. (2010). Generalized model learning for reinforcement learning on a humanoid robot. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2369–2374.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Holland, 1992] Holland, J. H. (1992). Complex adaptive systems. *Daedalus*, pages 17–30.

[Jaeger, 2008] Jaeger, H. (2008). A tutorial on training recurrent neural networks covering BPPT, RTRL, EKF, and the Echo-State Network approach. Technical report.

[Jakobi *et al.*, 1995] Jakobi, N., Husbands, P., and Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. *Advances in artificial life*, pages 704–720.

[Jiang and Zhai, 2007] Jiang, J. and Zhai, C. (2007). Instance weighting for domain adaptation in nlp. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, volume 7, pages 264–271, Prague, Czech Republic.

[Jones and Matarić, 2003] Jones, C. and Matarić, M. J. (2003). Adaptive division of labor in large-scale minimalist multi-robot systems. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, volume 2, pages 1969–1974.

[Kamien and Schwartz, 2012] Kamien, M. I. and Schwartz, N. L. (2012). *Dynamic optimization: the calculus of variations and optimal control in economics and management*. Courier Corporation.

[Karafotias *et al.*, 2011] Karafotias, G., Haasdijk, E., and Eiben, A. E. (2011). An algorithm for distributed on-line, on-board evolutionary robotics. In *Proc. of GECCO '11*, pages 171–178, New York. ACM.

[Kennedy, 2011] Kennedy, J. (2011). Particle swarm optimization. In *Encyclopedia of machine learning*, pages 760–766. Springer.

[Kernbach *et al.*, 2012] Kernbach, S., Nepomnyashchikh, V. A., Kancheva, T., and Kernbach, O. (2012). Specialization and generalization of robot behaviour in swarm energy foraging. *Mathematical and Computer Modelling of Dynamical Systems*, 18(1):131–152.

[Kernbach *et al.*, 2011] Kernbach, S., Schmickl, T., and Timmis, J. (2011). Collective adaptive systems: Challenges beyond evolvability. *CoRR*, abs/1108.5643.

[Kim *et al.*, 2004] Kim, H. J., Jordan, M. I., Sastry, S., and Ng, A. Y. (2004). Autonomous helicopter flight via reinforcement learning. In *Advances in neural information processing systems*, pages 799–806.

[Kirkpatrick *et al.*, 1983] Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., *et al.* (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

[Kitano, 1990] Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex systems*, 4(4):461–476.

[Kober and Peters, 2009] Kober, J. and Peters, J. R. (2009). Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, pages 849–856.

[Kohonen, 1998] Kohonen, T. (1998). The self-organizing map. *Neurocomputing*, 21(1):1–6.

[Konda and Tsitsiklis, 2000] Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014.

[Konig and Schmeck, 2009] Konig, L. and Schmeck, H. (2009). A completely evolvable genotype-phenotype mapping for evolutionary robotics. In *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 175–185.

[Koos *et al.*, 2012] Koos, S., Mouret, J.-B., and Doncieux, S. (2012). The Transferability Approach: Crossing the Reality Gap in Evolutionary Robotics. *IEEE Transactions on Evolutionary Computation*, pages 1–25.

[Kowaliw *et al.*, 2014] Kowaliw, T., Bredèche, N., and Doursat, R. (2014). *Growing Adaptive Machines: Combining Development and Learning in Artificial Neural Networks*. Springer Publishing Company, Incorporated.

[Kramer, 2010] Kramer, O. (2010). Evolutionary self-adaptation: a survey of operators and strategy parameters. *Evolutionary Intelligence*, 3(2):51–65.

[Labella *et al.*, 2006] Labella, T. H., Dorigo, M., and Deneubourg, J.-L. (2006). Division of labor in a group of robots inspired by ants' foraging behavior. *ACM Transactions on Autonomous and Adaptive Systems*, 1(1):4–25.

[Lamport, 1978] Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565.

[Lanzi *et al.*, 2003] Lanzi, P. L., Stolzmann, W., and Wilson, S. W. (2003). *Learning classifier systems: from foundations to applications*. Springer.

[Laureiro-Martínez *et al.*, 2015] Laureiro-Martínez, D., Brusoni, S., Canessa, N., and Zollo, M. (2015). Understanding the exploration–exploitation dilemma: An fMRI study of attention control and decision-making performance. *Strategic Management Journal*, 36(3):319–338.

[Lawrence and Platt, 2004] Lawrence, N. D. and Platt, J. C. (2004). Learning to learn with the informative vector machine. In *Proceedings of the twenty-first international conference on Machine learning*, page 65. ACM.

[Li *et al.*, 2004] Li, L., Martinoli, A., and Abu-Mostafa, Y. S. (2004). Learning and measuring specialization in collaborative swarm systems. *Adaptive Behavior*, 12(3-4):199–212.

[Liu and Winfield, 2010] Liu, W. and Winfield, A. F. (2010). Autonomous morphogenesis in self-assembling robots using ir-based sensing and local communications. In *ANTS Conference*, pages 107–118. Springer.

[Liu *et al.*, 2013] Liu, Y., Wu, A., Guo, D., Yao, K.-T., and Raghavendra, C. S. (2013). Weighted task regularization for multitask learning. In *Data Mining Workshops (ICDMW), 2013 IEEE 13th International Conference on*, pages 399–406. IEEE.

[Lloyd, 1982] Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137.

[López-Ibáñez *et al.*, 2011] López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., and Birattari, M. (2011). The irace package, iterated race for automatic algorithm configuration. Technical report, Université Libre de Bruxelles, Belgium.

[Luke *et al.*, 1998] Luke, S., Hohn, C., Farris, J., Jackson, G., and Hendler, J. (1998). Co-evolving soccer softbot team coordination with genetic programming. *RoboCup-97: Robot soccer world cup I*, pages 398–411.

[Maass, 1997] Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671.

[Mahfoud, 1995] Mahfoud, S. W. (1995). *Niching Methods for Genetic Algorithms*. PhD thesis, Champaign, IL, USA.

[Manner *et al.*, 1992] Manner, R., Mahfoud, S., and Mahfoud, S. W. (1992). Crowding and preselection revisited. In *Parallel Problem Solving From Nature*, pages 27–36. North-Holland.

[Maron and Moore, 1994] Maron, O. and Moore, A. W. (1994). Hoeffding races: Accelerating model selection search for classification and function approximation. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 59–66. Morgan-Kaufmann.

[Mattiussi and Floreano, 2007] Mattiussi, C. and Floreano, D. (2007). Analog genetic encoding for the evolution of circuits and networks. *Evolutionary Computation, IEEE Transactions on*, 11(5):596–607.

[Mauldin, 1984] Mauldin, M. L. (1984). Maintaining diversity in genetic search. In *AAAI*, pages 247–250.

[Mc Ginley *et al.*, 2011] Mc Ginley, B., Maher, J., O'Riordan, C., and Morgan, F. (2011). Maintaining healthy population diversity using adaptive crossover, mutation, and selection. *IEEE Transactions on Evolutionary Computation*, 15(5):692–714.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[McLurkin and Yamins, 2005] McLurkin, J. and Yamins, D. (2005). Dynamic task assignment in robot swarms. In *Robotics: Science and Systems*, volume 8. Cambridge, USA.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

[Mitchell *et al.*, 2015] Mitchell, T. M., Cohen, W. W., Talukdar, P. P., Betteridge, J., Carlson, A., Gardner, M., Kisiel, B., Krishnamurthy, J., *et al.* (2015). Never ending learning. In *AAAI*, pages 2302–2310.

[Mitchell and Thrun, 1993] Mitchell, T. M. and Thrun, S. B. (1993). Explanation-based neural network learning for robot control. In *Advances in neural information processing systems*, pages 287–294.

[Montanier and Bredèche, 2011a] Montanier, J.-M. and Bredèche, N. (2011a). Embedded evolutionary robotics: The (1+ 1)-restart-online adaptation algorithm. In *New horizons in evolutionary robotics*, pages 155–169. Springer.

[Montanier and Bredèche, 2011b] Montanier, J.-M. and Bredèche, N. (2011b). Emergence of altruism in open-ended evolution in a population of autonomous agents. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 25–26. ACM.

[Montanier and Bredèche, 2011c] Montanier, J.-M. and Bredèche, N. (2011c). Surviving the tragedy of commons: Emergence of altruism in a population of evolving autonomous agents. In *European Conference on Artificial Life*.

[Montanier and Bredèche, 2013] Montanier, J.-M. and Bredèche, N. (2013). Evolution of altruism and spatial dispersion: an artificial evolutionary ecology approach. In *European Conference on Artificial Life*, pages 260–267.

[Montanier *et al.*, 2016] Montanier, J.-M., Carrignon, S., and Bredèche, N. (2016). Behavioral specialization in embodied evolutionary robotics: Why so difficult? *Frontiers in Robotics and AI*, 3:38.

[Montes de Oca and Stützle, 2008] Montes de Oca, M. A. and Stützle, T. (2008). Towards incremental social learning in optimization and multiagent systems. In *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*, pages 1939–1944. ACM.

[Morimoto and Atkeson, 2003] Morimoto, J. and Atkeson, C. G. (2003). Minimax differential dynamic programming: An application to robust biped walking. In *Advances in neural information processing systems*, pages 1563–1570.

[Mouret, 2016] Mouret, J.-B. (2016). Micro-data learning: The other end of the spectrum. *ERCIM News*, (107):2.

[Mouret and Doncieux, 2009a] Mouret, J.-B. and Doncieux, S. (2009a). Evolving modular neural-networks through exaptation. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 1570–1577. IEEE.

[Mouret and Doncieux, 2009b] Mouret, J.-B. and Doncieux, S. (2009b). Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 1161–1168. IEEE.

[Murciano *et al.*, 1997] Murciano, A., Millán, J. d. R., and Zamora, J. (1997). Specialization in multi-agent systems through learning. *Biological cybernetics*, 76(5):375–382.

[Nelson *et al.*, 2009] Nelson, A. L., Barlow, G. J., and Doitsidis, L. (2009). Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345–370.

[Nguyen *et al.*, 2012] Nguyen, T. T., Yang, S., and Branke, J. (2012). Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, 6:1–24.

[Nitschke, 2005] Nitschke, G. (2005). Emergence of cooperation: State of the art. *Artificial Life*, 11(3):367–396.

[Nitschke *et al.*, 2012] Nitschke, G. S., Eiben, A., and Schut, M. C. (2012). Evolving team behaviors with specialization. *Genetic Programming and Evolvable Machines*, 13(4):493–536.

[Nolfi and Floreano, 2000] Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics*. MIT Press.

[Noskov *et al.*, 2013] Noskov, N., Haasdijk, E., Weel, B., and Eiben, A. (2013). Monee: Using parental investment to combine open-ended and task-driven evolution. In Esparcia-Alcázar, editor, *Applications of Evolutionary Computation*, volume 7835 of *LNCS*. Springer Berlin.

[Oliehoek and Amato, 2016] Oliehoek, F. A. and Amato, C. (2016). *A concise introduction to decentralized POMDPs*. Springer.

[Pan and Yang, 2010] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

[Peters *et al.*, 2010] Peters, J., Mülling, K., and Altun, Y. (2010). Relative entropy policy search. In *AAAI*, pages 1607–1612. Atlanta.

[Poirier and Silver, 2005] Poirier, R. and Silver, D. L. (2005). Effect of curriculum on the consolidation of neural network task knowledge. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 4, pages 2123–2128. IEEE.

[Prieto *et al.*, 2010] Prieto, A., Becerra, J., Bellas, F., and Duro, R. J. (2010). Open-ended evolution as a means to self-organize heterogeneous multi-robot systems in real time. *Robotics and Autonomous Systems*, 58(12):1282–1291.

[Pugh and Martinoli, 2006] Pugh, J. and Martinoli, A. (2006). Multi-robot learning with particle swarm optimization. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 441–448. ACM.

[Raina *et al.*, 2006] Raina, R., Ng, A. Y., and Koller, D. (2006). Constructing informative priors using transfer learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 713–720. ACM.

[Rasmussen and Williams, 2006] Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian processes for machine learning*, volume 1. MIT press Cambridge.

[Ratcliff, 1990] Ratcliff, R. (1990). Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285.

[Risi and Togelius, 2017] Risi, S. and Togelius, J. (2017). Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(1):25–41.

[Robins, 1995] Robins, A. (1995). Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146.

[Rokach and Maimon, 2005] Rokach, L. and Maimon, O. (2005). *Decision Trees*, pages 165–192. Springer US, Boston, MA.

[Rückstiess *et al.*, 2010] Rückstiess, T., Sehnke, F., Schaul, T., Wierstra, D., Sun, Y., and Schmidhuber, J. (2010). Exploring parameter space in reinforcement learning. *Paladyn*, 1(1):14–24.

[Rumelhart *et al.*, 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA.

[Rummery and Niranjan, 1994] Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering.

[Russell and Norvig, 2003] Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition.

[Ruvolo and Eaton, 2013a] Ruvolo, P. and Eaton, E. (2013a). Active task selection for lifelong machine learning. In *AAAI*.

[Ruvolo and Eaton, 2013b] Ruvolo, P. and Eaton, E. (2013b). Ella: An efficient lifelong learning algorithm. In *International Conference on Machine Learning*, pages 507–515.

[Şahin and Spears, 2004] Şahin, E. and Spears, W. M. (2004). *International Workshop Swarm Robotics at SAB 2004, Santa Monica, CA, USA, Revised Selected Papers*, volume 3342. Springer.

[Schwarzer *et al.*, 2012] Schwarzer, C., Schlachter, F., and Michiels, N. K. (2012). Online evolution in dynamic environments using neural networks in autonomous robots. *International Journal On Advances in Intelligent Systems*, 4(3 and 4):288–298.

[Sebag and Schoenauer, 1994] Sebag, M. and Schoenauer, M. (1994). Controlling crossover through inductive learning. *Parallel Problem Solving from Nature—PPSN III*, pages 209–218.

[Sehnke *et al.*, 2008] Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., and Schmidhuber, J. (2008). Policy gradients with parameter-based exploration for control. *Artificial Neural Networks-ICANN 2008*, pages 387–396.

[Shah *et al.*, 2009] Shah, D. *et al.* (2009). Gossip algorithms. *Foundations and Trends® in Networking*, 3(1):1–125.

[Sher, 2012] Sher, G. I. (2012). *Handbook of neuroevolution through Erlang*. Springer Science & Business Media.

[Siciliano and Khatib, 2016] Siciliano, B. and Khatib, O. (2016). *Springer handbook of robotics*. Springer.

[Silva *et al.*, 2014] Silva, F., Correia, L., and Christensen, A. L. (2014). Speeding up online evolution of robotic controllers with macro-neurons. In *European Conference on the Applications of Evolutionary Computation*, pages 765–776. Springer.

[Silva *et al.*, 2016] Silva, F., Correia, L., and Christensen, A. L. (2016). Online hyper-evolution of controllers in multirobot systems. In *2016 IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 11–20.

[Silva *et al.*, 2012a] Silva, F., Urbano, P., and Christensen, A. (2012a). Adaptation of robot behaviour through online evolution and neuromodulated learning. In Pavón, J., Duque-Méndez, N., and Fuentes-Fernández, R., editors, *Advances in Artificial Intelligence – IBERAMIA 2012*, volume 7637 of *Lecture Notes in Computer Science*, pages 300–309. Springer Berlin Heidelberg.

[Silva *et al.*, 2015] Silva, F., Urbano, P., Correia, L., and Christensen, A. L. (2015). odNEAT: An algorithm for decentralised online evolution of robotic controllers. *Evolutionary Computation*, 23(3):421–449.

[Silva *et al.*, 2012b] Silva, F., Urbano, P., Oliveira, S., and Christensen, A. L. (2012b). odneat: an algorithm for distributed online, onboard evolution of robot behaviours. In *Artificial Life*, volume 13, pages 251–258. MIT Press.

[Silver, 2013] Silver, D. L. (2013). The consolidation of task knowledge for lifelong machine learning. In *AAAI Spring Symposium: Lifelong Machine Learning*.

[Silver and McCracken, 2003] Silver, D. L. and McCracken, P. (2003). The consolidation of neural network task knowledge. In *Proceedings of the 2003 International Conference on Machine Learning and Applications - ICMLA 2003, June 23-24, 2003, Los Angeles, California, USA.*, pages 185–192.

[Silver and Poirier, 2004] Silver, D. L. and Poirier, R. (2004). Sequential consolidation of learned task knowledge. In *Canadian Conference on AI*, volume 3060, pages 217–232. Springer.

[Silver *et al.*, 2008] Silver, D. L., Poirier, R., and Currie, D. (2008). Inductive transfer with context-sensitive neural networks. *Machine Learning*, 73(3):313.

[Silver *et al.*, 2013] Silver, D. L., Yang, Q., and Li, L. (2013). Lifelong machine learning systems: Beyond learning algorithms. In *AAAI Spring Symposium: Lifelong Machine Learning*, pages 49–55. Citeseer.

[Smith *et al.*, 1997] Smith, P. J., Shafi, M., and Gao, H. (1997). Quick simulation: A review of importance sampling techniques in communications systems. *IEEE Journal on Selected Areas in Communications*, 15(4):597–613.

[Sperati *et al.*, 2010] Sperati, V., Trianni, V., and Nolfi, S. (2010). Evolution of self-organised path formation in a swarm of robots. In *ANTS Conference*, pages 155–166. Springer.

[Squillero and Tonda, 2016] Squillero, G. and Tonda, A. (2016). Divergence of character and premature convergence: A survey of methodologies for promoting diversity in evolutionary optimization. *Information Sciences*, 329:782 – 799. Special issue on Discovery Science.

[Stanley *et al.*, 2009] Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212.

[Stanley and Miikkulainen, 2002] Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127.

[Steyven *et al.*, 2016] Steyven, A., Hart, E., and Paechter, B. (2016). Understanding environmental influence in an open-ended evolutionary algorithm. In *International Conference on Parallel Problem Solving from Nature*, pages 921–931. Springer.

[Steyven *et al.*, 2017] Steyven, A., Hart, E., and Paechter, B. (2017). An investigation of environmental influence on the benefits of adaptation mechanisms in evolutionary swarm robotics. In *Proceedings of the Genetic and Evolutionary Computation Conference*, New York, NY, USA. ACM.

[Stone and Veloso, 2000] Stone, P. and Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383.

[Stradner *et al.*, 2009] Stradner, J., Hamann, H., Schmickl, T., Thenius, R., and Crailsheim, K. (2009). Evolving a novel bio-inspired controller in reconfigurable robots. In *European Conference on Artificial Life*, pages 132–139. Springer.

[Sutton and Barto, 1990] Sutton, R. S. and Barto, A. G. (1990). *Time-derivative models of pavlovian reinforcement.*, pages 497–537. The MIT Press, Cambridge, MA, USA.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

[Sutton *et al.*, 2000] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

[Tangkaratt *et al.*, 2014] Tangkaratt, V., Mori, S., Zhao, T., Morimoto, J., and Sugiyama, M. (2014). Model-based policy gradients with parameter-based exploration by least-squares conditional density estimation. *Neural networks*, 57:128–140.

[Taylor and Stone, 2009] Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685.

[Taylor and Stone, 2011] Taylor, M. E. and Stone, P. (2011). An introduction to intertask transfer for reinforcement learning. *Ai Magazine*, 32(1):15.

[Téllez and Angulo Bahón, 2008] Téllez, R. A. and Angulo Bahón, C. (2008). *Progressive design through staged evolution.* InTechOpen.

[Thrun and Mitchell, 1995] Thrun, S. and Mitchell, T. M. (1995). Lifelong robot learning. *Robotics and Autonomous Systems*, 15(1–2):25 – 46. The Biology and Technology of Intelligent Autonomous Agents.

[Thrun and Pratt, 1998] Thrun, S. and Pratt, L. (1998). Learning to learn: Introduction and overview. In *Learning to learn*, pages 3–17. Springer.

[Tomassini, 2005] Tomassini, M. (2005). *Spatially-structured evolutionary algorithms.* Springer Berlin.

[Torrey and Shavlik, 2009] Torrey, L. and Shavlik, J. (2009). Transfer learning. *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, 1:242.

[Torrey *et al.*, 2008] Torrey, L., Shavlik, J., Natarajan, S., Kuppili, P., and Walker, T. (2008). Transfer in reinforcement learning via markov logic networks. In *AAAI Workshop on Transfer Learning for Complex Tasks*.

[Touzet, 1997] Touzet, C. F. (1997). Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems*, 22(3-4):251–281.

[Trianni, 2008] Trianni, V. (2008). *Evolutionary Swarm Robotics: Evolving Self-Organising Behaviours in Groups of Autonomous Robots (Studies in Computational Intelligence)*. Springer, 1 edition.

[Trianni and Nolfi, 2009a] Trianni, V. and Nolfi, S. (2009a). Evolving collective control, cooperation and distributed cognition. *Handbook of Collective Robotics*, pages 127–166.

[Trianni and Nolfi, 2009b] Trianni, V. and Nolfi, S. (2009b). Self-organizing sync in a robotic swarm: a dynamical system view. *IEEE Transactions on Evolutionary Computation*, 13(4):722–741.

[Trianni *et al.*, 2014] Trianni, V., Tuci, E., Ampatzis, C., and Dorigo, M. (2014). Evolutionary swarm robotics: A theoretical and methodological itinerary from individual neuro-controllers to collective behaviours. *The Horizons of Evolutionary Robotics*, 153.

[Trueba *et al.*, 2013] Trueba, P., Prieto, A., Bellas, F., Caamaño, P., and Duro, R. J. (2013). Specialization analysis of embodied evolution for robotic collective tasks. *Robotics and Autonomous Systems*, 61(7):682–693.

[Tuci, 2014] Tuci, E. (2014). *Evolutionary Swarm Robotics: Genetic Diversity, Task-Allocation and Task-Switching*, pages 98–109. Springer International Publishing, Cham.

[Tuci *et al.*, 2008] Tuci, E., Ampatzis, C., Vicentini, F., and Dorigo, M. (2008). Evolving homogeneous neurocontrollers for a group of heterogeneous robots: Coordinated motion, cooperation, and acoustic communication. *Artificial Life*, 14(2):157–178.

[Vassilev *et al.*, 2003] Vassilev, V. K., Fogarty, T. C., and Miller, J. F. (2003). *Smoothness, Ruggedness and Neutrality of Fitness Landscapes: from Theory to Application*, pages 3–44. Springer, Berlin, Heidelberg.

[Waibel *et al.*, 2006] Waibel, M., Floreano, D., Magnenat, S., and Keller, L. (2006). Division of labour and colony efficiency in social insects: effects of interactions between genetic architecture, colony kin structure and rate of perturbations. *Proceedings of the Royal Society of London B: Biological Sciences*, 273(1595):1815–1823.

[Waibel *et al.*, 2009] Waibel, M., Keller, L., and Floreano, D. (2009). Genetic team composition and level of selection in the evolution of cooperation. *Transactions on Evolutionary Computation*, 13(3):648–660.

[Walker *et al.*, 2006] Walker, J. H., Garrett, S. M., and Wilson, M. S. (2006). The balance between initial training and lifelong adaptation in evolving robot controllers. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 36(2):423–432.

[Wang *et al.*, 2016] Wang, D., Wang, H., and Liu, L. (2016). Unknown environment exploration of multi-robot system with the FORDPSO. *Swarm and Evolutionary Computation*, 26:157–174.

[Watkins, 1989] Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, King's College, Cambridge.

[Watson *et al.*, 2002] Watson, R. A., Ficici, S. G., and Pollack, J. B. (2002). Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Syst.* Elsevier.

[Weiss, 1999] Weiss, G. (1999). *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press.

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

[Wischmann *et al.*, 2007] Wischmann, S., Stamm, K., and Wörgötter, F. (2007). Embodied evolution and learning: The neglected timing of maturation. *Advances in Artificial Life*, pages 284–293.

[Yamaguchi and Atkeson, 2016] Yamaguchi, A. and Atkeson, C. G. (2016). Neural networks and differential dynamic programming for reinforcement learning problems. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5434–5441.

[Yang, 2005] Yang, S. (2005). Memory-based immigrants for genetic algorithms in dynamic environments. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1115–1122. ACM.

[Yang, 2008] Yang, X.-S. (2008). *Nature-Inspired Metaheuristic Algorithms*. Luniver Press.

[Yosinski *et al.*, 2011] Yosinski, J., Clune, J., Hidalgo, D., Nguyen, S., Zagal, J. C., and Lipson, H. (2011). Evolving robot gaits in hardware: the hyperneat generative encoding vs. parameter optimization. In *Proceedings of the European Conference in Artificial Life*.

[Yu and Suganthan, 2009] Yu, E. and Suganthan, P. N. (2009). Evolutionary programming with ensemble of explicit memories for dynamic optimization. In *IEEE Congress on Evolutionary Computation (CEC'09)*, pages 431–438. IEEE.

[Zagal and Ruiz-Del-Solar, 2007] Zagal, J. C. and Ruiz-Del-Solar, J. (2007). Combining simulation and reality in evolutionary robotics. *Journal of Intelligent Robotics Systems*, 50(1):19–39.

[Zimmer and Doncieux, 2017] Zimmer, M. and Doncieux, S. (2017). Bootstrapping q-learning for robotics from neuro-evolution results. *IEEE Transactions on Cognitive and Developmental Systems*, PP(99).

List of Abbreviations

`irace` Iterated Race for Automatic Algorithm Configuration. 92

ACO Ant Colony Optimization. 20

AE Artificial Evolution. 27, 29

AGE Analog Gene Encoding. 34, 36

ANN Artificial Neural Network. 7, 10, 12, 16, 17, 23, 25, 26, 33, 34, 36, 43, 45, 47, 52, 57, 58, 74, 84, 85, 102, 104, 105, 109, 115

CAP Credit-Assignment Problem. 9, 20, 39

CAS Collective Adaptive System. 4, 113

CPPN Compositional Pattern-Producing Network. 36

CTRNN Continuous-Time Recurrent Neural Network. 33

Dec-POMDP Decentralized Partially-Observable Markov Decision Process. 20, 21

DOP Dynamic Optimization Problem. 16, 26, 102, 103

EA Evolutionary Algorithm. 5, 7, 11, 12, 16, 20, 21, 26, 29–33, 35, 37–43, 46, 55–62, 69, 72, 82–85, 101–103, 105, 109, 113–115

EDEA Embodied Distributed Evolutionary Algorithm. 62

EDO Evolutionary Dynamic Optimization. 16, 26, 27, 57, 101–103

EE Embodied Evolution. 5, 27, 29, 38, 40–47, 51, 52, 55–58, 72–74, 82, 84–88, 99, 113, 114, 117

EER Embodied Evolutionary Robotics. 6, 7, 10–12, 21, 22, 27, 37, 38, 40, 42, 43, 45–47, 51, 57–60, 62–65, 68, 69, 71, 73, 82–86, 101, 109, 110, 113–118

ER Evolutionary Robotics. 10, 11, 16, 21, 27, 29–33, 36, 40, 41, 47, 53, 55, 58, 60–63, 69, 71, 82, 90, 113, 117

ES Evolution Strategies. 35, 43, 45

ESN Echo-State Network. 33

ESR Evolutionary Swarm Robotics. 29, 37–39, 47, 56, 71

FAEA Forgetting-Avoidance Evolutionary Algorithm. 12, 58, 101, 103, 104, 109

GA Genetic Algorithm. 36

GC Gene Clock. 58, 84, 86, 88–90, 93, 97, 99, 100

Hyper-NEAT Hypercube-based NeuroEvolution of Augmenting Topologies. 36

LC Logical Clock. 86, 87

LML Lifelong Machine Learning. 16, 24–27, 110

LSTM Long Short-Term Memory. 33

MAS Multiagent System. 7, 9, 43

mEDEA minimal Environment-driven Distributed Evolutionary Adaptation. 44, 45, 52, 58, 62–64, 68, 73–75, 82, 142

ML Machine Learning. 5, 7, 10, 15–18, 21–27, 30, 57, 102, 113

MONEE Multi-Objective aNd open-Ended Evolution. 44, 45, 63, 64

MTL Multi-Task Learning. 23–25

NEAT NeuroEvolution of Augmenting Topologies. 36, 47, 62, 83–86, 88, 89, 116

odNEAT online distributed NeuroEvolution of Augmenting Topologies. 47, 58, 62, 84–89, 91, 92, 99, 116

OHE Online Hyper-Evolution. 47

PGPE Policy Gradients with Parameter-based Exploration. 20

PGTA Probabilistic Gene Transfer Algorithm. 43, 62

PSO Particle Swarm Optimization. 20, 21

RL Reinforcement Learning. 10, 15, 16, 18–20, 24, 25, 27, 46, 60, 113

SI Swarm Intelligence. 4, 7, 113

SL Supervised Learning. 17, 18, 23, 26

SR Swarm Robotics. 4, 37

TL Transfer Learning. 23, 24

TWEANN Topology-and-Weights Evolving Artificial Neural Network. 34

UL Unsupervised Learning. 17, 18

Résumé

Cette thèse concerne l'adaptation par apprentissage évolutionnaire de comportements dans des essaims d'agents robotiques. Les essaims naturels montrent une grande capacité d'adaptation à leurs environnements, ce qui fait augmenter le succès reproductif des individus dans les essaims. Concrètement, le sujet d'étude de cette thèse se trouve au croissement entre les systèmes complexes et l'apprentissage automatique. Les systèmes complexes sont des systèmes composés d'un grand nombre d'unités qui interagissent entre elles et avec leur environnement, ce qui résulte en l'émergence de comportements globaux. L'apprentissage automatique concerne l'amélioration progressive d'un système artificiel à partir des données. Concrètement, cette thèse présente des contributions liées à l'adaptation ou apprentissage automatique de comportements dans des essaims d'agents robotiques, qui sont des sytèmes complexes composés d'un grand nombre de robots en interaction.

Cette adaptation est réalisée par le moyen d'algorithmes d'évolution artificielle (*Artificial Evolution*), qui sont inspirés de l'évolution darwinienne. Plus précisément, nous utilisons des algorithmes d'évolution incarnée et distribuée (*distributed Embodied Evolutionary Robotics, dEER*), dans lesquels chaque robot de l'essaim exécute une instance d'algorithme évolutionnaire pendant qu'il opère dans son environnement, et il échange des expériences de l'apprentissage avec les robots qu'il rencontre. Malgré le fait que ces algorithmes présentent différents avantages (*e.g.* parallelisation et partage de l'apprentissage, adaptation en ligne, *i.e.* pendant que l'essaim opère, etc ...), ils ont été introduits relativement récemment, et certaines de leurs propriétés restent à étudier. Dans cette thèse, nous présentons un ensemble de contributions qui permettent de mieux comprendre et maîtriser les algorithmes de *dEER*.

Distributed Embodied Evolutionary Robotics

Un tel algorithme (*cf.* Figure 9.1) fonctionne de la façon suivante. Chaque robot dans l'essaim exécute par lui même (*onboard*) une instance d'algorithme évolutionnaire. Il initialise de manière aléatoire son contrôleur (un réseau de neurones dans nos études). Le robot exécute ce contrôleur au fur et à mesure qu'il opère dans son environnemnt, et, en utilisant une récompense fournie par l'environnement, il maintient à tout moment une estimation de sa performance par rapport à la tâche qui lui est affectée, ou *fitness*. Lorsque deux robots se rencontrent, ils échangent leurs respectifs contrôleurs et valeurs de *fitness*, qui sont ensuite stockés dans des listes locales à chaque robot, ou populations locales. Après un certain temps d'exécution d'un robot et échange génétique avec d'autres robots, l'agent robotique s'arrête, et charge le nouveau contrôleur à évaluer afin d'optimiser le comportement par rapport à la *fitness*. Pour choisir quel contrôleur charger, le robot sélectionne un contrôleur parent dans sa liste locale (*sélection*), qui est ensuite légèrement modifié de manière aléatoire afin d'explorer de nouveaux comportements (*mutation*). Le contrôleur résultant de cette mutation remplace ensuite le contrôleur précédent, et une nouvelle itération

de l'algorithme démarre.

Algorithm 9.1: Structure basique d'un algorithme de *distributed Embodied Evolutionary Robotics* Il correspond à mEDEA [Bredèche *et al.*, 2012] avec une pression de sélection dirigée vers la tâche, et il inclut de l'autoinsemination et le vidange générationnel de la population locale.

```
1  g_a ← random()
2  while true do
3      l ← ∅, f ← 0
4      for t ← 1 to T_e do
5          exec(g_a)
6          f ← evaluate()
7          broadcast(g_a, f)
8          l ← l ⋃ listen()
9      l ← l ⋃ {(g_a, f)}
10     selected ← select(l)
11     g_a ← mutate(selected)
```

Dans cette thèse, nous étudions comment adapter automatiquement des comportements dans des essaims de robots par le biais des algorithmes de *dEER*. Nos contributions apportent un éclaircissement sur différents aspects du fonctionnement de ces méthodes. Concrètement, ces contributions se focalisent sur certains composants algorithmiques en *dEER* et comment ces méthodes se comportent dans différents environnements: l'impact de la pression à la sélection sur les populations locales de chaque agent, l'évolution distribuée de la structure des neurocontrôleurs des agents dans l'essaim, et l'adaptation distribuée à la collaboration. Des perspectives d'ouverture sont discutées à la fin du document. Nous décrivons maintenant l'ensemble des contributions et les perspectives.

Contributions

Les trois contributions majeures de cette thèse dans le context des algorithmes d'évolution incarnée et distribuée concernent l'impact de la *pression à la sélection* sur les populations locales, la *évolution de la structure* du neurocontrôleur, et *l'adaptation à la collaboration*, qui sont décrites ensuite :

Pression à la sélection

La pression à la sélection dans un système évolutionnaire dénote n'importe quel cause qui impacte la survie des individus du système. Typiquement, en robotique évolutionnaire, la valeur de *fitness* des individus dans la population est utilisée par l'algorithme d'adaptation pour sélectionner progressivement des contrôleurs mieux adaptés au critère défini. Classiquement, la pression à la sélection est légèrement diminuée pour permettre à l'algorithme d'explorer son espace de recherche, et réguler l'équilibre entre exploration pure et exploitation pure.

Par contre, dans le cas distribué de la *dEER* la sélection est réalisée sur des populations restraintes : les populations locales de chaque robot dans l'essaim, qui sont composées des contrôleurs récoltés lors des rencontres entre robots. De ce fait, la pression à la sélection est appliquée

différemment, et est donc susceptible d'avoir un comportement différent vis-à-vis de la propagation des contrôleurs performants. Afin de mieux comprendre le rôle de la pression à la sélection dans les algorithmes de *dEER*, nous réalisons une série d'expériences dans lesquelles différents niveaux de pression à la sélection sont appliqués dans un essaim de robots qui s'adapte à deux différents tâches : navigation avec évitement d'obstacles et fourragement. Dans les deux tâches, nous comparons les niveaux de pression à la sélection par rapport à un ensemble de critères de performance, et nous concluons que, dans les conditions évaluées, il y a une tendance généralisée selon laquelle une pression à la sélection maximale fournit les meilleures performances de toutes les variantes, ce qui n'est pas le cas dans la robotique évolutionnaire classique (centralisée).

Évolution de la structure

La bonne adaptation des contrôleurs des robots dans un essaim dépend en grande mesure de la représentation choisie pour les contrôleurs (des neurocontrôleurs dans nos expériences), ainsi que des opérateurs de mutation, qui explorent l'espace de ces contrôleurs. Pour ce faire, les opérateurs de mutation modifient progressivement les contrôleurs sélectionnés. Dans les expériences sur la pression à la sélection, les neurocontrôleurs ont une structure fixe, et la mutation modifie ses paramètres. Si l'on souhaite augmenter l'expressivité des réseaux de neurones, il est possible de faire grandir leur structure : il existe des algorithmes évolutionnaires qui font évoluer la structure et les poids des réseaux de neurones simultanément. Par contre, dans le cas distribué, ceci nécessite d'identifier uniquement et pouvoir ordonner dans le temps les neurones et connections des contrôleurs dans l'essaim.

Pour répondre à ce besoin, nous proposons et évaluons *Gene Clocks (GC)* une méthode complètement décentralisée pour tracer historiquement les éléments des réseaux de neurones lors de la neuroévolution de structure et paramètres. Nous évaluons notre méthode dans deux tâches similaires aux expériences précédentes, pour tester si notre méthode proposée permet de faire évoluer la structure et paramètres des réseaux de neurones dans un essaim de robots. Nos expériences montrent que notre méthode, qui est complètement décentralisée, évolue la structure et paramètres des neurocontrôleurs sans nuire la performance par rapport à une variante où les éléments neuronaux sont identifiés et ordonnés de manière parfaite en utilisant des informations globales, qui, typiquement, ne sont pas disponibles dans un essaim de robots.

Adaptation à la collaboration

Les individus dans les essaims naturels montrent des comportements collaboratifs qui leur permettent d'améliorer leur efficacité et de résoudre des problèmes de plus grande envergure. Dans le cas des essaims de robots, l'adaptation progressive pour la résolution d'un problème intrinsèquement collaboratif reste un défi difficile en général. Les algorithmes d'adaptation centralisée fournissent des conditions plus propices pour que la collaboration puisse émerger; pourtant, dans le cadre distribué des algorithmes de *dEER*, cela reste un défi majeur.

Afin d'évaluer la capacité des algorithmes de *dEER* pour adapter les comportements d'un essaim de robots pour qu'ils collaborent, nous définissons une tâche intrinsèquement collaborative, *i.e.* une tâche qui ne peut pas être résolue individuellement : la récolte collaborative d'objets, où les robots doivent récolter des objets, et deux robots doivent se coordonner pour récolter chaque objet. Nous réalisons une série d'expériences où un essaim de robots doit s'adapter pour réaliser cette tâche, et nous analysons en détail les résultats. Nous concluons que l'essaim réussit à s'adapter pour collaborer, en grande partie grâce à la capacité des robots de trouver et approcher de manière jointe les objets. Néanmoins, nous avons identifié certaines inefficacités au niveau

de la coordination des robots et de la synchronisation de leurs actions. Ces inefficacités sont amoindries lorsqu'une pression de sélection maximale est appliquée.

Perspectives

Les études présentées dans cette thèse ont aussi ouvert des questions de recherche comme perspectives qui mériteraient d'être investiguées. Nous listons les deux perspectives les plus importantes, qui sont décrites plus en détail dans le dernier chapître du document.

Apprentissage incrémental : les algorithmes de *dEER* permettent l'apprentissage en ligne des comportements d'un essaim de robots. Ceci implique l'exploration et adaptation continue à l'environnement et aux tâches, qui sont susceptibles de changer. Dans le cas des changements, il pourrait être bénéfique pour les robots de cumuler incrémentalement leur expérience, retenir ce qui a été appris, et l'exploiter pour mieux s'adapter dans le futur. Ceci est connu sous le nom de *Lifelong Learning*, et reste une des questions majeures dans le domaine de l'apprentissage automatique. Nous estimons que réaliser un tel apprentissage de manière efficace constituerait une avancée majeur dans le domaine de la robotique en essaim et du *Machine Learning*.

Tâches émérgentes : les tâches employées dans les expériences dans cette thèse restent des tâches relativement simples, et des tâches soit individuelles, soit nécessitant une collaboration à deux robots. Pourtant, dans la nature et dans le domaine du *Swarm Intelligence*, souvent les comportements individuels et interactions des unités résultent en un comportement émergent qui résout la tâche de manière globale. L'évolution distribuée de comportements pour de telles tâches est un défi extrèmement compliqué, et nous estimons que des algorithmes capables d'adapter de tels comportements augmenterait significativement l'applicabilité des robots en essaim.

Les contributions scientifiques de cette thèse ont éclaircit le fonctionnemment des algorithmes de *dEER* par rapport à différentes questions cruciales lorsqu'on souhaite adapter automatiquement des comportements dans des essaims de robots. Ceci montre que les algorithmes de *distributed Embodied Evolutionary Robotics* constituent un outil puissant pour l'adaptation de comportements dans des essaims d'agents robotiques.

Abstract

Robot swarms are systems composed of a large number of rather simple robots. Due to the large number of units, these systems, have good properties concerning robustness and scalability, among others. However, it remains generally difficult to design controllers for such robotic systems, particularly due to the complexity of inter-robot interactions. Consequently, automatic approaches to synthesize behavior in robot swarms are a compelling alternative. One of these approaches, Embodied Evolutionary Robotics (EER), opens many possibilities, due to learning taking place in parallel for each robot in the swarm, while deployed for task operation, *i.e.* online. Parallel evaluations and information exchanges among robots accelerate learning, which is open-ended, thus allowing for potential adaptation to changing conditions. That said, EER approaches are relatively new, and their properties remain to be studied.

In this thesis, we focus on online behavior adaptation in a swarm of robots using distributed EER methods. We consider a swarm of robots that coexist in an environment, and must progressively adapt to given tasks. Additionally, since robots may face changing conditions that may repeat over time, retaining acquired knowledge about previous conditions could improve their adaptivity. However, when confronted to new situations, adaptive systems may instantaneously forget what was learned before, thus hindering such adaptivity. The contributions in this thesis aim at investigating and improving the adaptivity of evolving robot swarms. To this end, we provide four main contributions:

- We investigate the influence of task-driven selection pressure in a swarm of robotic agents using a distributed EER approach. We evaluate the impact of a range of selection operators on the performance of a distributed EER algorithm for a robot swarm. The results show that task-driven selection pressure is necessary when addressing given tasks in such a distributed setup, and the higher the selection pressure, the better the performances obtained.

- We investigate the evolution of collaborative behaviors in a swarm of robotic agents using a distributed EER approach. We perform a set of experiments for a swarm of robots to adapt to a collaborative item collection task that cannot be solved by a single robot. Our results show that the swarm learns to collaborate to solve the task using a distributed approach. Additionally, some inefficiencies regarding learning to choose actions to collect items are analyzed, and perspectives are discussed to improve action choice.

- We propose and experimentally validate a completely distributed mechanism that allows to learn the structure and parameters of the robot neurocontrollers in a swarm using a distributed EER approach. This allows for the robot controllers to augment their capacity and expressivity. Our experiments show that our fully-decentralized mechanism leads to similar results as a mechanism that depends on global information.

- We propose an algorithm to avoid forgetting from the perspective of an evolving population when adapting to changing conditions. In a set of preliminary experiments, we test a centralized version of the algorithm, showing the feasibility of the approach. Finally, we discuss how it can be transferred to the decentralized context of distributed EER.

Keywords: Evolutionary Swarm Robotics, Online Adaptation, Neuroevolution

Résumé

Les essaims de robots sont des systèmes composés d'un grand nombre de robots relativement simples. Du fait du grand nombre d'unités, ces systèmes ont de bonnes propriétés de robustesse et de passage à l'échelle. Néanmoins, il reste en général difficile de concevoir manuellement des contrôleurs pour les essaims de robots, à cause de la grande complexité des interactions inter-robot. Par conséquent, les approches automatisées pour l'apprentissage de comportements d'essaims de robots constituent une alternative attrayante. Une de ces approches, *Embodied Evolutionary Robotics (EER)*, ouvre de nombreuses possibilités, puisque l'apprentissage est mené en parallèle par chaque robot de l'essaim et en ligne pendant l'exécution de la tâche. Grâce à cette parallélisation et aux échanges d'information entre les robots, l'apprentissage est accéléré, et, en outre, vu que l'apprentissage a lieu en continu, l'adaptation à des conditions variables devient possible.

Dans cette thèse, nous étudions l'adaptation de comportements d'essaim de robots avec des méthodes de *EER* distribuée. Nous considérons un essaim de robots dans un environnement, qui doivent s'adapter progressivement pour réaliser les tâches qui leur sont affectées. Vu que les robots peuvent confronter différentes tâches au cours du temps, et qu'elles peuvent se répéter, retenir des comportements efficaces afin de les exécuter lorsqu'elles reapparaîtront pourrait améliorer l'adaptativité de l'essaim. Par contre, quand ils sont confrontés à des nouvelles tâches, les systèmes adaptatifs sont susceptibles d'oublier de manière immédiate ce qu'ils ont appris en amont, ce qui nuit cette adaptativité. Les contributions dans cette thèse visent à investiguer et améliorer l'adaptativité des essaims de robots. Ainsi, nous fournissons quatre contributions principales:

- Nous étudions l'influence de la pression à la sélection dirigée vers une tâche dans un essaim d'agents robotiques qui utilisent une approche d'*EER* distribuée. Nous évaluons l'impact de différents opérateurs de sélection dans un algorithme d'*EER* distribuée pour un essaim de robots. Nos résultats montrent que la sélection est nécessaire lorsque les robots doivent s'adapter à des tâches particulières. De plus, lorsque la pression à la sélection est plus forte, les performances sont meilleures.
- Nous étudions l'évolution de comportements collaboratifs pour une tâche de récolte d'objets dans un essaim d'agents robotiques qui utilisent une approche d'*EER* distribuée. Nous réalisons un ensemble d'expériences où un essaim de robots s'adapte à une tâche collaborative avec un algorithme d'*EER* distribuée. Nos résultats montrent que l'essaim s'adapte à résoudre la tâche. Des limitations concernant l'apprentissage du choix d'action pour récolter des objets sont identifiées, et nous présentons des perspectives pour l'améliorer.
- Nous proposons et validons par des expériences un mécanisme complètement distribué qui permet d'adapter la structure des neurocontrôleurs des robots dans un essaim qui utilise une approche d'*EER* distribuée. Ceci permet aux contrôleurs des robots d'augmenter leur expressivité et leur capacité de stockage. Nos expériences montrent que notre mécanisme, qui est complètement décentralisé, fournit des résultats similaires à un mécanisme qui dépend d'une information globale.
- Nous proposons un algorithme pour éviter l'oubli selon la perspective d'une population qui s'adapte à des conditions qui changent. Nous réalisons des expériences préliminaires avec une version centralisée de l'algorithme, et nos résultats montrent la faisabilité de l'approche. Enfin, nous discutons comment adapter l'approche au contexte décentralisé de l'*EER* distribuée.

Mots-clés: Robotique Évolutionaire en Essaim, Adaptation en Ligne, Neuroévolution

Abstract

Robot swarms are systems composed of a large number of rather simple robots. Due to the large number of units, these systems, have good properties concerning robustness and scalability, among others. However, it remains generally difficult to design controllers for such robotic systems, particularly due to the complexity of inter-robot interactions. Consequently, automatic approaches to synthesize behavior in robot swarms are a compelling alternative. In this thesis, we focus on online behavior adaptation in a swarm of robots using distributed Embodied Evolutionary Robotics (EER) methods. To this end, we provide three main contributions:

- We investigate the influence of task-driven selection pressure in a swarm of robotic agents using a distributed EER approach. We evaluate the impact of a range of selection pressure strength on the performance of a distributed EER algorithm. The results show that the stronger the task-driven selection pressure, the better the performances obtained when addressing given tasks.
- We investigate the evolution of collaborative behaviors in a swarm of robotic agents using a distributed EER approach. We perform a set of experiments for a swarm of robots to adapt to a collaborative item collection task that cannot be solved by a single robot. Our results show that the swarm learns to collaborate to solve the task using a distributed approach, and we identify some inefficiencies regarding learning to choose actions.
- We propose and experimentally validate a completely distributed mechanism that allows to learn the structure and parameters of the robot neurocontrollers in a swarm using a distributed EER approach, which allows for the robot controllers to augment their expressivity. Our experiments show that our fully-decentralized mechanism leads to similar results as a mechanism that depends on global information.

Keywords: Evolutionary Swarm Robotics, Online Adaptation, Neuroevolution

Résumé

Les essaims de robots sont des systèmes composés d'un grand nombre de robots relativement simples. Du fait du grand nombre d'unités, ces systèmes ont de bonnes propriétés de robustesse et de passage à l'échelle. Néanmoins, il reste en général difficile de concevoir manuellement des contrôleurs pour les essaims de robots, à cause de la grande complexité des interactions inter-robot. Par conséquent, les approches automatisées pour l'apprentissage de comportements d'essaims de robots constituent une alternative attrayante. Dans cette thèse, nous étudions l'adaptation de comportements d'essaim de robots avec des méthodes de *Embodied Evolutionary Robotics (EER)* distribuée. Ainsi, nous fournissons trois contributions principales:

- Nous étudions l'influence de la pression à la sélection dirigée vers une tâche dans un essaim d'agents robotiques qui utilisent une approche d'*EER* distribuée. Nous évaluons l'impact de différents opérateurs de sélection dans un algorithme d'*EER* distribuée pour un essaim de robots. Nos résultats montrent que le plus forte la pression à la sélection est, les meilleures performances sont atteintes lorsque les robots doivent s'adapter à des tâches particulières.
- Nous étudions l'évolution de comportements collaboratifs pour une tâche de récolte d'objets dans un essaim d'agents robotiques qui utilisent une approche d'*EER* distribuée. Nous réalisons un ensemble d'expériences où un essaim de robots s'adapte à une tâche collaborative avec un algorithme d'*EER* distribuée. Nos résultats montrent que l'essaim s'adapte à résoudre la tâche, et nous identifions des limitations concernant le choix d'action.
- Nous proposons et validons experimentalement un mécanisme complètement distribué pour adapter la structure des neurocontrôleurs des robots dans un essaim qui utilise une approche d'*EER* distribuée, ce qui permettrait aux neurocontrôleurs d'augmenter leur expressivité. Nos expériences montrent que notre mécanisme, qui est complètement décentralisé, fournit des résultats similaires à un mécanisme qui dépend d'une information globale.

Mots-clés: Robotique Évolutionaire en Essaim, Adaptation en Ligne, Neuroévolution