UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

UNIVERSITÀ DEGLI STUDI DI TRIESTE

XXXVII CICLO DEL DOTTORATO DI RICERCA IN

APPLIED DATA SCIENCE AND ARTIFICIAL INTELLIGENCE

Towards Bio-Inspired Interpretable Embodied Artificial Intelligence

Settore scientifico-disciplinare: **IINF-05/A** (ex **ING-INF/05**)

DOTTORANDO / A
GIORGIA NADIZAR

COORDINATORE
PROF. FRANCESCO PAULI

SUPERVISORE DI TESI
PROF. ERIC MEDVET

CO-SUPERVISORE DI TESI
PROF. STEFANO NICHELE

ANNO ACCADEMICO 2023/2024

Towards Bio-Inspired Interpretable Embodied Artificial Intelligence

Merging Bio-Mimicry with Transparency in Robot Controllers

Ph.D. Dissertation

by

Giorgia Nadizar

Master of Science in Computer Engineering, University of Trieste, Italy

Giorgia Nadizar
Towards Bio-Inspired Interpretable Embodied Artificial Intelligence
Supervisor: Prof. Eric Medvet
Co-Supervisor: Prof. Stefano Nichele

University of Trieste
Department of Mathematics, Informatics and Geosciences
Ph.D. Program in Applied Data Science & Artificial Intelligence
Academic Year 2023/2024

Any sufficiently advanced technology is indistinguishable from magic.
Arthur C. Clarke

To bicycles and planes,
my primary sources of magic

Abstract

Embodied Artificial Intelligence (AI) refers to the integration of AI systems within a (simulated) body, such as AI-controlled robots. This paradigm is inspired by biological organisms, aiming to replicate the dynamic interaction between intelligent behavior and a body. However, the resemblance to biological beings is frequently a coarse approximation, raising questions about the extent to which these systems genuinely capture biological principles. In addition, AI-controlled robots bring about concerns related to trust and transparency, particularly due to the inherent difficulty in understanding the underlying decision-making processes of these agents. In fact, people tend to trust AI systems more when they are transparent and interpretable. However, many AI models applied in robotic control, e.g., Artificial Neural Networks (ANNs), lack the necessary transparency, which makes them harder to trust.

In this work, we focus on these two interrelated aspects of embodied AI: the enhancement of biological resemblance in AI-controlled robots and the improvement of their interpretability. We investigate these issues in the context of Modular Soft Robots (MSRs), which serve as an ideal test bed due to their inherent similarity to biological organisms. The first part of this work centers on increasing the biological fidelity in these embodied AI agents by experimenting with different ANN models that aim to mimic neural processes observed in nature, and by simulating biological phenomena, such as neural and morphological plasticity. In the second part of the study, we turn to the issue of interpretability, a key factor in ensuring that AI-controlled robots are trustworthy and reliable. Due to the highly subjective nature of interpretability, we first investigate the factors that contribute to the individual perceived interpretability of AI systems. We then design interpretable controllers for the robotic domain, starting with benchmark case studies involving rigid robots. Finally, we merge the two axes of investigation—biological resemblance and interpretability—by devising interpretable controllers for biologically inspired MSRs and comparing them against more complex neural controllers.

Our results show that (1) bio-inspiration can enhance the performance of embodied agents, (2) interpretability does not need to compromise performance, and (3) bio-inspiration and interpretability are not mutually exclusive, indicating that it is feasible to pursue a path *towards bio-inspired interpretable embodied AI*.

Contents

1

Introduction

The term Artificial Intelligence (AI) was coined in 1956 during the Dartmouth Summer Research Project [254] to capture the idea of machines capable of *human-like* information processing. Since then, the term AI has become more and more of an umbrella term, capturing disparate and increasingly advanced examples of automatic information processing conducted by machines [243], ranging from automatic image recognition [418], to complex game playing [280, 390], passing through chat-bots [323] and digital assistants [249]. As such, AI has become an integral part of most people's daily lives, often leading to the misconception that it represents a *real* form of intelligence due to its name—with many people even questioning its *sentience* [381].

Although this constitutes an exacerbation (and a clear misinterpretation) of the term, what is true is that AI was originally inspired by studies on the functioning of biological brains. Even before the concept acquired its current name, McCulloch and Pitts [255] coarsely emulated the biological neuron in the artificial, studying how logical functions could be implemented through it. After their trailblazer work, a multitude of studies followed along, with two main identifiable intents: (1) gaining better understanding on biological processes underlying the emergence of biological intelligence, or (2) achieving improved information processing capabilities on the artificial side.

The field of robotics has witnessed a similar progression, starting from the early depictions of artificial beings in R.U.R. ("Rossum's Universal Robots") by Čapek [56]—which introduced the term "robot"—and continuing through the sci-fi stories of Asimov [14], to the actual applications of AI on robots [209]. In this domain, the term *embodied AI* refers to the application of the concept of

embodiment to AI, where AI is integrated within a physical (or simulated) robotic agent [63, 335], i.e., an agent with a *body*. The embodiment allows the AI agent to interact with the environment, perceive itself, and perform tasks in real-time, much like living entities.

This enhances the potential for linking AI with its biological counterpart, as biological intelligence initially developed within a body. In fact, embodied AI can play a pivotal role in the development of more accurate models of biological organisms, thereby deepening our understanding of biological intelligence [77]. Aligned with this goal, the embodied Turing test challenges AI models to interact with the sensorimotor world at skill levels akin to their living counterparts [470]. These embodied AI models could simulate complex behaviors and processes observed in nature, providing valuable insights into how living organisms process information and respond to their surroundings.

On the other hand, embodiment enables robotic agents to perform various tasks thanks to a suitably trained AI-driven controller [119], which acts for the robot as a form of *brain* [402]. Moreover, the generalization abilities of AI could endow the robot with an unprecedented autonomy, which could allow to adapt to changing environments and scenarios where more classical robotic control techniques might fail [265, 184].

Remarkably, these two aspects of embodied AI are not mutually exclusive. Indeed, they find ideal convergence in the paradigm of *Modular Soft Robots (MSRs)*, a class of robots composed of soft and elastic modules [321, 153]. The structure of MSRs draws inspiration from the muscular tissue of living beings, where the modules can contract and expand similarly to muscular cells: this makes them particularly suitable for reproducing lifelike behaviors in silico [105, 423]. Additionally, softness and modularity grant an unparalleled degree of adaptability. While softness confers MSRs with compliance—allowing for interaction with fragile material [411], locomotion on rugged terrains [453], or navigation in tight spaces [62]—modularity enables vast design freedom [5], automatic assembly [455], and reconfiguration [288].

In this setting, one important consideration is that in deploying autonomous AI agents in the wild, performance is not the only significant aspect. It is equally relevant to *interpret* (and possibly foresee) the decisions made by these agents to ensure they operate safely [8]. In fact, transparency in decision-making processes is essential to trust and reliability, particularly in critical applications as those involving autonomous robots [386].

A possible path towards interpretability in robotics is realizing simple and concise controllers, where their constrained size allows for some form of interpretability via direct human inspection [222]. Such a strategy is viable when the considered system, i.e., robot, is simple enough, yet it becomes more challenging and might require further enhancements when dealing with more complex

agents, as the MSRs mentioned before. Namely, MSRs display two elements of complexity w.r.t. more standard paradigms in robotics, e.g., rigid robots. First, their modular nature sees an ensemble of interdependent components, which require coordination and synergy [51]. Second, the softness of their modules determines a non-trivial body dynamics, which could exacerbate the difficulties of finding an effective controller for them [233]. Thus, finding a MSR controller that is both effective and interpretable constitutes an arduous challenge.

In summary, the interplay between biological inspiration and interpretability represents a fundamental tension within the field of embodied AI. Biological systems, as complex and often inscrutable entities, offer invaluable insights into the development of more advanced, adaptive, and lifelike AI systems. The rich dynamics of biological processes provide a blueprint for novel approaches to both robotic control and AI-driven behaviors. However, the very complexity that makes biological systems so powerful also presents a challenge to interpretability. In contrast, interpretability in AI strives for simplicity and transparency, ensuring that the actions and decisions of autonomous agents can be understood, trusted, and predicted.

Given these premises, a complex scenario emerges, where different and potentially contrasting questions arise (we formalize them in Section 1.1). This thesis is uniquely positioned to explore the convergence of these two themes—biological inspiration and interpretability—investigating how we can draw from the intricacies of biological models while maintaining a level of interpretability necessary for ensuring trustworthiness in practical applications. In this thesis, we identify key elements from both domains, and examine how to combine them to design autonomous systems that are both high-performing and comprehensible.

1.1 Research Questions

We hereby formalize the research questions guiding this thesis. We address them through a sequence of experimental analyses, further detailed in the remainder of the thesis.

1.1.1 Bio-Inspiration in Embodied AI

How can the integration of biological principles into AI for robotics enhance the performance and autonomy of embodied agents, and to what extent do in-silico findings reflect and illuminate phenomena in the biological domain?

This research question further unfolds in the following more specific research questions, which we address in Part I.

Research Question 1.1 *How do different models of Artificial Neural Networks (ANNs) perform as controllers for embodied agents? Namely, are more biologically plausible models more effective/efficient/general? Do they induce different behaviors?*

We consider this question in Chapters 3 and 4.

In Chapter 3 we conduct a thorough experimental comparison of different models of ANNs for controlling MSRs. We investigate the differences along various axes, as effectiveness, efficiency, generalization ability, and also induced behavior. The chapter is based on: **G. Nadizar**, E. Medvet, S. Nichele, and S. Pontes-Filho. An experimental comparison of evolved neural network models for controlling simulated modular soft robots. *Applied Soft Computing*, page 110610, 2023 [299].

In Chapter 4 we also consider the interaction between different types of ANNs and collective intelligence, proposing a paradigm of collective control for MSRs based on bio-inspired models of ANNs. We assess the performance of the proposed framework in terms of effectiveness and adaptability, examining also some samples of behaviors. The chapter is based on: **G. Nadizar**, E. Medvet, S. Nichele, and S. Pontes-Filho. Collective control of modular soft robots via embodied Spiking Neural Cellular Automata. In *Workshop on From Cells to Societies: Collective Learning across Scales (Part of ICLR)*, 2022 [297].

Research Question 1.2 *Can we replicate the biological phenomenon of plasticity in embodied agents? Do embodied agents benefit from plasticity? In what terms do they differ from non plastic ones?*

We address this question in Chapters 5 and 6, considering brain plasticity and morphological plasticity, respectively.

In Chapter 5 we combine body-brain co-optimization of MSRs with brain plasticity in the form of Hebbian learning. Not only do we measure the quantitative differences in performance w.r.t. non plastic controllers, but we also study if Hebbian learning induces qualitative differences among the modular controllers (i.e., some form of specialization). The chapter is based on: A. Ferigo*, G. Iacca*, E. Medvet*, and **G. Nadizar*** (* equal contribution). Totipotent Neural Controllers for Modular Soft Robots: Achieving Specialization in Body-Brain Co-Evolution through Hebbian Learning. *Neurocomputing*. 2024 [113].

In Chapter 6, instead, we consider morphological plasticity, i.e., development. First, we provide a proof-of-concept on how it is possible to simulate development for MSRs. Then, we investigate how the morphological development of these agents should be scheduled to obtain high-performing autonomous agents, comparing them against non developing ones as baseline. The chapter is based on: **G. Nadizar**, E. Medvet, and K. Miras. On the Schedule for Morphological Development of Evolved Modular Soft Robots. In *European Conference on Genetic*

Programming (Part of EvoStar), pages 146–161. Springer, 2022 [296].

Research Question 1.3 *Is it possible to leverage the biological phenomenon of synaptic pruning to reduce over-parameterization in the ANNs used to control embodied agents? Does pruning hinder the performance or does it promote adaptability to unforeseen circumstances?*

We address this question in Chapter 7, where we study the effects of synaptic pruning in MSRs. We consider different pruning schemes from the literature and assess their impact at different pruning rates on the performance and on the adaptability of the agents. The chapter is based on: **G. Nadizar**, E. Medvet, O. H. Ramstad, S. Nichele, F. A. Pellegrino, and M. Zullich. Merging pruning and neuroevolution: towards robust and efficient controllers for modular soft robots. *The Knowledge Engineering Review*, 37, 2022 [298].

1.1.2 Interpretability in Embodied AI

How can we achieve interpretability in the control of embodied agents without sacrificing performance and considering the subjective nature of this notion?

This question opens the research line towards interpretability, to which we dedicate Part II. We present the more specific questions guiding our study in the following.

Research Question 2.1 *Can we exploit a human-in-the-loop system to discover AI models that are both well-performing and interpretable for a given user? What elements are needed for such a system to be most effective?*

We explore this question in Chapter 8, generalizing an existing framework for personalized interpretability estimation with human-in-the-loop. We validate our proposal on both simulated and real users, testing its sensitivity w.r.t. different users behaviors and notions of interpretability. The chapter is based on: **G. Nadizar***, L. Rovito*, A. De Lorenzo, E. Medvet, and M. Virgolin (* shared first co-author). An analysis of the ingredients for learning interpretable symbolic regression models with human-in-the-loop and genetic programming. *ACM Transactions on Evolutionary Learning*, 2024 [303].

Research Question 2.2 *Can we leverage graph optimization techniques to find graph control policies that are both interpretable and effective? Do we need to explicitly promote interpretability or can it emerge naturally? Is diversity a suitable avenue for this goal?*

We tackle this question in Chapters 9 and 10.

In Chapter 9 we apply two graph optimization techniques on several continuous control tasks. We compare the results against relevant non-interpretable

baselines for performance, and against the results of a search directed by interpretability. The chapter is based on: **G. Nadizar**, E. Medvet, and D. G. Wilson. Naturally interpretable control policies via graph-based genetic programming. In *European Conference on Genetic Programming (Part of EvoStar)*, pages 73–89. Springer, 2024 [302].

In Chapter 10 we propose a framework to diversify graph control policies both in terms of behavior and structure. We validate it assessing performance, diversity, interpretability, and robustness against unforeseen damages. The chapter is based on: **G. Nadizar**, E. Medvet, and D. Wilson. Searching for a Diversity of Interpretable Graph Control Policies. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 933–941, 2024 [301].

1.1.3 Integrating Bio-Inspiration and Interpretability in Embodied AI

Can we identify a paradigm of bio-inspired interpretable embodied AI that balances the complexity and the benefits of bio-mimicry with the need for transparency for trustworthiness in control?

This is the last and most comprehensive research question of the thesis, where all previous investigations converge into a unified framework. It centers on the crucial theme of integrating interpretability with biological systems: while biological systems are inherently complex and difficult to understand, interpretability seeks to create simple and comprehensible models. The key question is whether these two objectives can be harmonized. It unfolds in two more precise research questions, addressed in Part III.

Research Question 3.1 *Is it feasible to rely on more compact controllers for MSRs? What approach is the most effective for their optimization between direct optimization and imitation learning?*

We tackle this question in Chapter 11, where we compare different controller representations for MSRs, with various degrees of transparency. We assess and compare the controllers effectiveness upon direct optimization and upon imitation learning. The chapter is based on: E. Medvet and **G. Nadizar**. GP for Continuous Control: Teacher or Learner? The Case of Simulated Modular Soft Robots. In *Genetic Programming Theory and Practice XX*, pages 203–224. Springer, 2024 [257].

Research Question 3.2 *Can diversity foster the optimization of MSRs that are interpretable, effective, and adaptable? How important is it to diversify in terms of body designs, controllers features, and robot behaviors?*

We address this question in Chapter 12, by optimizing MSRs with a focus on their body, controller, and behavior diversity, considering both opaque and interpretable controllers. We compare the results obtained, assessing the impact of diversity on performance and adaptability of the evolved agents.

1.2 Summary of Contributions

By addressing the research questions detailed in Section 1.1 the contributions of the thesis are the following.

1. First, in Part I, it explores the *integration of several biological principles in the artificial domain*, studying them on MSRs. In this context, we highlight several similarities between artificial agents (and AI) and biological organisms. Specifically, we show that ANNs can yield better performance at controlling embodied agents when displaying higher biological resemblance. We also note that embodied AI agents can benefit from plasticity processes. In addition, we observe that pruning in ANNs can reduce the controller parameters without affecting the performance.

2. Second, in Part II, it moves in the direction of *introducing interpretability in the domain of robotics*. As a first step, we focus on the subjective nature of interpretability, improving and generalizing an existing human-in-the-loop framework which is able to find the most interpretable AI models for a given user. Subsequently, we introduce interpretable controllers for rigid non-modular robots, often finding state-of-the-art performance with transparent graph-based policies.

3. Last, in Part III, it aims at *merging the bio-inspiration aspect with the interpretability one*, by proposing ways of crafting interpretable controllers for MSRs. Namely, we show how to design and optimize symbolic and graph-based controllers for MSRs, leveraging diversity as a tool for yielding better performance. Also in this case we note minor performance degradation w.r.t. opaque ANN-based controllers.

Before presenting the three core contributions in Parts I to III, we provide a background chapter that lays the foundation for understanding the thesis. Specifically, in Chapter 2, we introduce MSRs, ANNs, and Evolutionary Computation (EC), from which we derive the optimization techniques used throughout the thesis. Finally, in Part IV, we summarize the key findings, answer the research questions, and discuss the broader impact and future directions of this work.

2

Background

Prior to delving into the research questions and the experimental analyses conducted to address them, we dedicate this chapter to providing the reader with the needed background to fully grasp the remainder of the manuscript. In further detail, we present the paradigm of MSRs in Section 2.1, giving some notions about existing models—both real and simulated—and providing a detailed description of those employed. Then, we move towards the AI aspects, giving an overview of different types of ANNs and how they can be employed as robotic controllers in Section 2.2. Last, we conclude the chapter with a presentation of EC, from which we take the techniques we leverage for robot optimization.

2.1 Modular Soft Robots

Biological organisms exhibit *modularity* at various levels [241]—from the cellular level to the emergence of complex bodies and brains without any form of centralized control. Additionally, these organisms are *soft*, displaying different degrees of compliance with respect to their surroundings—again at various scales.

In the artificial domain, modular robotics [5] provides a framework for the investigations of biologically inspired principles of collective control and coordination [61]. Moreover, modular robots can be highly reconfigurable and capable of self-assembly [329], providing fault tolerance and reusability of modules. Furthermore, incorporating the element of softness not only enhances the bio-inspired component [67], but also increases adaptability and applicability across various domains [20, 404]. Thus, MSRs represent the ideal convergence of these paradigms.

Modular robotics offers significant potential *advantages*, such as low cost through mass fabrication of modules, adaptability by enabling changes of configuration, and robustness by allowing the replacement of only damaged parts. However, several *challenges* persist. On the physical side, realizing individual units, i.e., modules, and ensuring effective connectivity among them are non-trivial tasks [475]. On the simulation side, accurately modeling the behavior and interactions of modular soft systems remains a significant hurdle [233].

MSRs encompass multiple categories, as surveyed by Zhang et al. [475], differing across various aspects. First, their assembly methods vary: some need to be pre-assembled, others are reconfigurable, and some are even self-reconfigurable. Additionally, diverse actuation mechanisms exist, including bending, elongation, inflation, shrinking, and twisting. This variety also extends to their sensing and actuation capabilities, as well as their connection methods, providing a wide range of possibilities.

In this scenario, Voxel-based Soft Robots (VSRs) [153]—composed of identical cubic modules—represent an appealing trade-off between functionality, ease of realization, and ease of simulation. In fact, the uniform cubic shape of the modules simplifies both their physical construction and their simulation, including the connectivity aspect. However, despite their uniform shape, these modules can possess different physical properties and actuation mechanisms, enhancing design possibilities and functionalities.

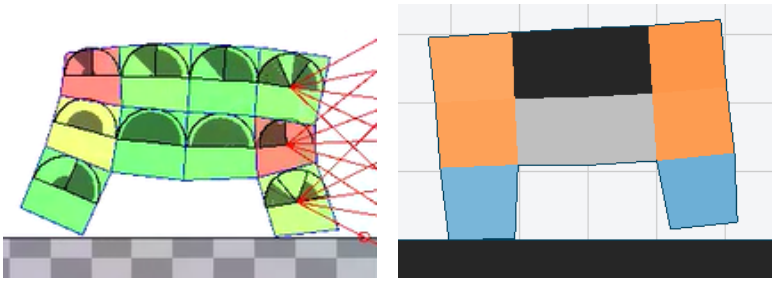
Concerning their physical realization, Hiller and Lipson [153] initially proposed a realization of VSRs featuring modules made of silicon foam with passive actuation. More recent developments, such as those by Kriegman et al. [208] and Sui et al. [408], relied on the same material but incorporating pneumatic actuation. In contrast, Legrand et al. [218] introduced a multi-material approach based on Diels-Alder polymers, enabling a wide range of mechanical properties along with reconfigurability. Additionally, some visionary approaches, such as that of Kriegman et al. [207], have explored the use of living matter.

Regarding the simulation aspect, there exist three main simulators for VSRs: (1) VoxCAD [154], which offers a 3D simulation, and (2) 2D-VSR-Sim [261] and (3) Evolution Gym (EvoGym) [32], which both provide a 2D simulation environment.

In this manuscript, we focus on VSRs as MSRs, and thus, we will use these terms interchangeably in the following. Moreover, we consider only a simulated 2D version of these robots, relying on 2D-VSR-Sim or EvoGym—see Figure 2.1 for a graphical representation of 2D VSRs. We use 2D-VSR-Sim in Chapters 3 to 7 and 11, while we employ EvoGym in Chapter 12.

Both simulators provide a simulation in discrete time and continuous space. For 2D-VSR-Sim, $f_{\text{sim}} = 60\,\text{Hz}$, whereas for EvoGym there are two frequencies to consider: the simulation update frequency, $f_{\text{sim}} = 1500\,\text{Hz}$, and the maximum

control frequency $f_{\text{c,max}} = 50\,\text{Hz}$. We provide more details about the mechanical model of the simulated VSRs in Section 2.1.1, while we devote Sections 2.1.2 and 2.1.3 to a description of their morphology and controller, respectively.

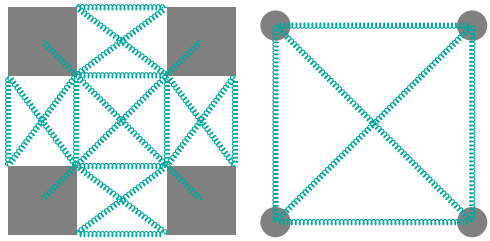


(a) 2D-VSR-Sim. (b) EvoGym.

Figure 2.1: Frames of the same VSR, a biped, in the two simulators, 2D-VSR-Sim and EvoGym. In 2D-VSR-Sim (Figure 2.1a) the color of each voxel encodes the ratio between its current area and its rest area: red indicates contraction, yellow rest state, and green expansion. In addition, the circular sector drawn at the center of each voxel indicates the current sensed values: sub-sectors represent different sensors within the same voxel; the rays of the proximity sensors are shown in red. In EvoGym (Figure 2.1b), the color of each voxel indicates its type (vertically active, horizontally active, soft passive, rigid).

2.1.1 Mechanical Model of the Voxel

2D-VSR-Sim and EvoGym model voxels in a similar, yet not identical, manner. Both simulators represent voxels as squares, where adjacent voxels are welded together, not allowing relative displacement. Four rigid masses delimit the corners of each voxel—square masses in 2D-VSR-Sim and point masses in EvoGym—and are joined together by ideal springs, as shown in Figure 2.2. 2D-VSR-Sim relies on spring-damper systems [87] characterized by a *frequency* f, in Hz, describing how many oscillations the spring executes per second, and a damping ratio d, dimensionless, describing how rapidly oscillations decay over time. Conversely, EvoGym uses ideal springs [308] with a Hooke coefficient k_j (in N/m), which depends on the type of the voxel j and on the spring placement (main springs are placed around the edges, whereas structural springs are placed on the cross-brace). Namely, there are four types of voxels in EvoGym, visually represented with different colors: (1) passive rigid voxels, (2) passive soft voxels, (3) horizontally active voxels, and (4) vertically active voxels.



(a) 2D-VSR-Sim. (b) EvoGym.

Figure 2.2: The mechanical models of the voxel in the two simulators. Squares and circles are rigid bodies, while wiggly strings are springs.

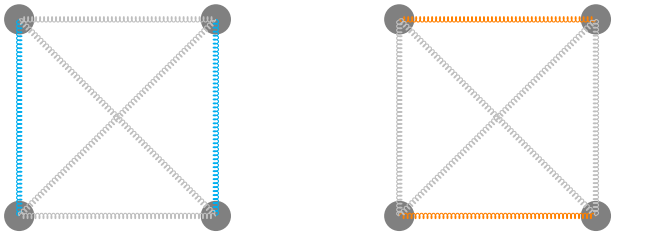
A VSR agent acts by changing the area of the composing voxels over time. In the mechanical model of 2D-VSR-Sim, actuation happens by varying the resting length of springs: given a control signal $a \in [-1, 1]$ for a voxel of side l, the resting length of the voxel springs changes instantaneously such that the voxel side becomes $l' = \sqrt{l^2(1 - a\rho_a)}$, with $\rho_a > 0$ being the *active range* (i.e., the maximum rate of increase or decrease of the voxel area). Note that the control signal a causes the rest length of the springs, not the actual length, to change instantaneously: the actual length changes gradually depending on f and d.

In EvoGym, actuation occurs similarly, i.e., by varying the lengths of the springs, but only the main vertical/horizontal springs of vertically/horizontally active voxels (■/■) can be actively controlled (see Figure 2.3). In this case the control signal $a \in [0.6, 1.6]$ changes the equilibrium length of the spring in a smooth manner, by computing a weighted average between the target length (weighted by α) and the current length (weighted by $1 - \alpha$). Moreover, EvoGym implements strain limiting, by repositioning masses if any spring grows/shrinks by more than $25\,\%$ for soft/active voxels (■, ■, ■) and $3\,\%$ for rigid voxels (■).

We summarize the parameters of both simulators and their values in Table 2.1. The experiments presented in the remainder of the manuscript employ the default values, unless otherwise specified.

2.1.2 Morphology

The morphology of a VSR describes the shape and the type of voxels in its body and its sensory apparatus. The *shape* of a VSR is defined as a two-dimensional grid of voxels. More formally, it can be defined as a polyomino, i.e., a plane geometric figure formed by joining one or more equal squares edge to edge. This definition descends from the fact that all voxels are modeled as ideal squares, which can easily be arranged in a grid. Adjacent voxels in the grid are joined by



(a) Vertically active voxel. (b) Horizontally active voxel.

Figure 2.3: The mechanical model of the active voxels in EvoGym: the colored springs are the *active* ones, i.e., those that can be controlled with the control signal, whereas the light gray ones are the *passive* ones, i.e., those only subjected to external forces.

	Symbol	Name	Value
2D-VSR-Sim		control signal range	$[-1, 1]$
	f_{sim}	simulation frequency	$60\,\text{Hz}$
		sensors domain	$[0, 1]$
	f	spring frequency	$8\,\text{Hz}$
	d	damping ratio	0.3
	ρ_a	active range	0.2
	μ_k	sliding friction	10
EvoGym		control signal range	$[0.6, 1.6]$
		sensors domain	$]-\infty, \infty[$
	f_{sim}	simulation frequency	$1500\,\text{Hz}$
	$f_{\text{c,max}}$	maximum control frequency	$50\,\text{Hz}$
		rigid main spring constant	$8.57 \times 10^7\,\text{N/m}$
		rigid structural spring constant	$4.29 \times 10^7\,\text{N/m}$
		soft main spring constant	$6 \times 10^7\,\text{N/m}$
		soft structural spring constant	$3 \times 10^7\,\text{N/m}$
		actuator main spring constant	$5 \times 10^7\,\text{N/m}$
		actuator structural spring constant	$1.25 \times 10^7\,\text{N/m}$
		maximum soft/active strain	$25\,\%$
		maximum rigid strain	$3\,\%$

Table 2.1: Main parameters (with default values) of the mechanical model of the voxel in the two considered simulators.

welding together the masses at their corners to form the overall robot.

We remark that it is also possible for voxels to self-assembly and reconfigure [368], although we do not investigate this aspect in this manuscript. In that case, the individual voxels are free to attach and detach, even resulting in a VSR shape different than a grid.

Moving on to the *sensory apparatus* of VSRs, we define it as the compound of sensors available to the VSR and their placement along the VSR body. Sensors can be used by the VSR for proprioception and to perceive various aspects of the surrounding environment. At each simulation time step k, each j-th sensor outputs a reading $r_j^{(k)}$. The domain of the sensors differs in the two simulators employed: $r_j^{(k)} \in [0, 1]$ in 2D-VSR-Sim, while $r_j^{(k)} \in]-\infty, \infty[$ in EvoGym.

- Area sensors sense the ratio between the current area of the voxel they are placed in and its rest area.
- Touch sensors perceive if a voxel is in contact with the ground, $r_j^{(k)} = 1$, or not, $r_j^{(k)} = 0$.
- Velocity sensors sense the velocity of the voxel center of mass along the voxel x- or y-axes.
- Proximity sensors perceive the distance towards objects along a predefined direction α. The sensed value corresponds to the distance between the voxel center of mass and the closest object. In case no object is detected below a distance threshold τ_{prox}, the corresponding reading is set to τ_{prox}.

In 2D-VSR-Sim, for area, velocity, and proximity sensors we employ a soft normalization of the readings using the $\tanh$ function and rescaling, to ensure each reading lies in $[0, 1]$. Conversely, we employ the raw readings in EvoGym.

2.1.3 Controller

The VSR controller is responsible for computing the control signals for all active voxels at every time step. The controller can be a simple open loop controller or a more complex closed loop controller. The former can be considered as a parametric function, whereas the latter can be considered as a parametric dynamical system.

Phase Controller

In the phase controller [153, 65, 204], each control signal is computed from the current time, according to a sinusoidal function. Namely, the control signal of

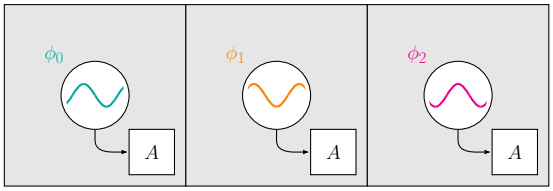


Figure 2.4: A schematic representation of the phase controller controller for a 3×1 VSR. Each voxel has its own phase ϕ_i. The boxes with the letter A represent the actuators.

the i-th voxel at simulation time step k is computed as

$$a_i^{(k)} = f_{\phi_i}(k) = \sin\left(2\frac{f_{\text{sin}}}{f_c}\pi k + \phi_i\right), \tag{2.1}$$

where f_{sin} is the sine wave frequency, f_c is the control frequency (possibly equal to the simulation frequency f_{sim}), and ϕ_i is the voxel phase. In most works where they have been used, these controllers have been optimized only in the phases $\boldsymbol{\phi}$, whereas f_{sin} is set a priori to the same value for each voxel: thus, these are called phase controllers. This is an open loop controller as it does not take into account the sensor readings for computing the control signal. We show a schematic representation of the phase controller for a 3×1 VSR in Figure 2.4.

Centralized Controller

In the centralized controller there is a *centralized* dynamical system processing the sensory information coming from each voxel to compute all the control signals of the VSR. At each time step k, this controller processes the concatenation $\boldsymbol{r}^{(k)} \in \mathbb{R}^{|\boldsymbol{r}|} = \begin{bmatrix}\boldsymbol{r}_0^{(k)} & \boldsymbol{r}_1^{(k)} & \dots\end{bmatrix}$ of the current sensor readings to update its state $\boldsymbol{s}^{(k)} \in \mathbb{R}^{|\boldsymbol{s}|}$ and compute the control signals for the n active voxels $\boldsymbol{a}^{(k)} \in \mathbb{R}^n$. Namely, the centralized controller can be described by the following set of equations:

$$\boldsymbol{s}^{(k)} = g_{\boldsymbol{\theta}}\left(\boldsymbol{r}^{(k)}, \boldsymbol{s}^{(k-1)}\right), \tag{2.2}$$

$$\boldsymbol{a}^{(k)} = f_{\boldsymbol{\theta}}\left(\boldsymbol{r}^{(k)}, \boldsymbol{s}^{(k)}\right), \tag{2.3}$$

$$\boldsymbol{s}^{(0)} = \boldsymbol{s}_0, \tag{2.4}$$

where $g_{\boldsymbol{\theta}} : \mathbb{R}^{|\boldsymbol{r}|} \times \mathbb{R}^{|\boldsymbol{s}|} \to \mathbb{R}^{|\boldsymbol{s}|}$ is the state update function, $f_{\boldsymbol{\theta}} : \mathbb{R}^{|\boldsymbol{r}|} \times \mathbb{R}^{|\boldsymbol{s}|} \to \mathbb{R}^n$ is the output function, and $\boldsymbol{s}_0 \in \mathbb{R}^{|\boldsymbol{s}|}$ is the initial state. Both state update and output functions depend on some parameters $\boldsymbol{\theta}$.

This controller is often implemented by an ANN [409], although other implementations are also possible. In addition, in various cases this controller becomes a simple parametric function rather than a proper dynamical system—i.e., it has no state. We show a schematic representation of the centralized controller for a 3×1 VSR in Figure 2.5.

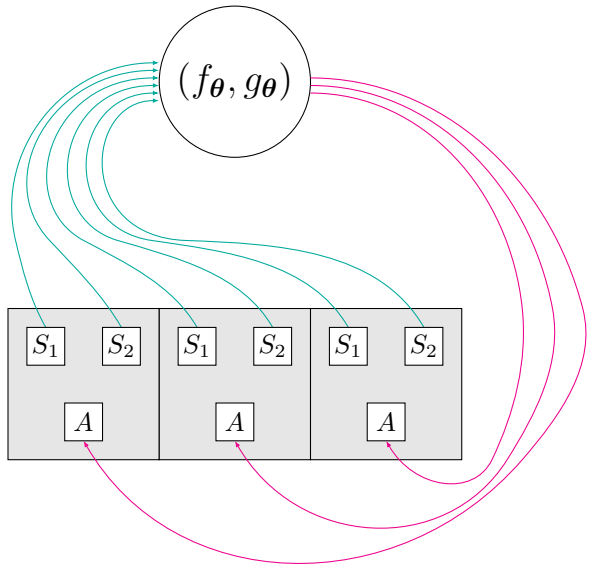


Figure 2.5: A schematic representation of the centralized controller for a 3×1 VSR with two sensors in each voxel. Green and magenta curved arrows represent the connection of the controller with inputs (sensors) and outputs (actuators), respectively.

Distributed Controller

The distributed, or modular, or embodied, controller architecture exploits the intrinsic modularity of VSRs [267]. The key idea is that each voxel is equipped with a local controller, which processes local inputs to produce the actuation value for said voxel. To allow coordination, the local controller can also access the sensor readings of all its neighbors, according to the Moore neighborhood, i.e., its neighbors on the sides and diagonals.

At every time step k, each local controller, i.e., dynamical system, computes

the actuation value for the i-th voxel according to the following set of equations:

$$\boldsymbol{s}_i^{(k)} = g_{i,\boldsymbol{\theta}_i}\left(\left[\boldsymbol{r}_i^{(k)}\ \boldsymbol{r}_{i,\nwarrow}^{(k)}\ \boldsymbol{r}_{i,\uparrow}^{(k)}\ \boldsymbol{r}_{i,\nearrow}^{(k)}\ \boldsymbol{r}_{i,\rightarrow}^{(k)}\ \boldsymbol{r}_{i,\searrow}^{(k)}\ \boldsymbol{r}_{i,\downarrow}^{(k)}\ \boldsymbol{r}_{i,\swarrow}^{(k)}\ \boldsymbol{r}_{i,\leftarrow}^{(k)}\right], \boldsymbol{s}_i^{(k-1)}\right), \tag{2.5}$$

$$a_i^{(k)} = f_{i,\boldsymbol{\theta}_i}\left(\left[\boldsymbol{r}_i^{(k)}\ \boldsymbol{r}_{i,\nwarrow}^{(k)}\ \boldsymbol{r}_{i,\uparrow}^{(k)}\ \boldsymbol{r}_{i,\nearrow}^{(k)}\ \boldsymbol{r}_{i,\rightarrow}^{(k)}\ \boldsymbol{r}_{i,\searrow}^{(k)}\ \boldsymbol{r}_{i,\downarrow}^{(k)}\ \boldsymbol{r}_{i,\swarrow}^{(k)}\ \boldsymbol{r}_{i,\leftarrow}^{(k)}\right], \boldsymbol{s}_i^{(k)}\right), \tag{2.6}$$

$$\boldsymbol{s}_i^{(0)} = \boldsymbol{s}_{i,0}, \tag{2.7}$$

where $\boldsymbol{r}_i^{(k)} \in \mathbb{R}^{|\boldsymbol{r}_i|}$ are the local sensor readings, all $\boldsymbol{r}_{i,\bullet}^{(k)} \in \mathbb{R}^{|\boldsymbol{r}_{i,\bullet}|}$ terms are the sensor readings of the Moore neighbors, listed in clockwise order starting from the top left, $g_{i,\boldsymbol{\theta}_i} : \mathbb{R}^{|\boldsymbol{s}_i|} \times \mathbb{R}^{|\boldsymbol{r}_i|+|\boldsymbol{r}_{i,\nwarrow}|+\cdots+|\boldsymbol{r}_{i,\leftarrow}|} \to \mathbb{R}^{|\boldsymbol{s}_i|}$ is the local state update function, $f_{i,\boldsymbol{\theta}_i} : \mathbb{R}^{|\boldsymbol{s}_i|} \times \mathbb{R}^{|\boldsymbol{r}_i|+|\boldsymbol{r}_{i,\nwarrow}|+\cdots+|\boldsymbol{r}_{i,\leftarrow}|} \to \mathbb{R}$ is the local output function, and $\boldsymbol{s}_{i,0} \in \mathbb{R}^{|\boldsymbol{s}_i|}$ is the local initial state. All state update and output functions depend on some parameters $\boldsymbol{\theta}_i$.

If the voxel has no neighbor on a given direction two solutions are possible: either no information is passed, or the controller is fed with a $\mathbf{0}$ vector. The second strategy is important in relation to the following observation. The functions $f_{i,\boldsymbol{\theta}_i}$ and $g_{i,\boldsymbol{\theta}_i}$ are in principle all different, but it is also possible to have the same function, i.e., the same dynamical system, in all voxels, provided some constraints about input sizes are satisfied. Namely, if all functions are the same, all voxels need to have the same amount of sensors, and if there is no neighbor on a given side a $\mathbf{0}$ vector of the proper dimension is required. As for the centralized controller, the distributed controller can become a simple parametric function rather than a proper dynamical system in cases with no state.

In Figure 2.6, we show a schematic representation of the distributed controller for a 2×2 VSR with 3 voxels, in which we ignore the inputs coming from non-existing voxels.

Distributed Controller with Explicit Communication

A more sophisticated distributed controller version is that of Medvet et al. [260], where adjacent voxels are connected by means of n_c *communication* channels, to enable the transfer of information along the body of the VSR. Thus, each dynamical system reads the local sensors values together with the $4n_c$ values coming from adjacent voxels (belonging to the Von Neumann neighborhood). In turn, the output function computes the actuation signal and $4n_c$ values to feed to contiguous voxels—the 4 deriving by the fact that communication is *directional*, i.e., the communication values are dedicated for each neighbor. Note that this controller architecture results in an overall recurrence across the VSR body, which is responsible for introducing an additional dynamics to the one deriving from the mechanical model of the VSR.

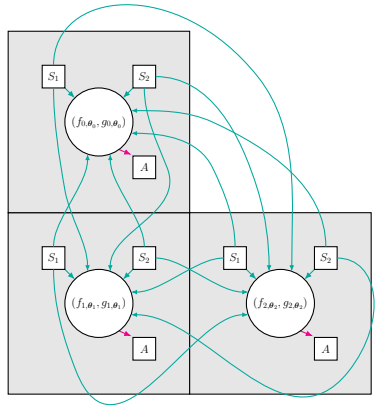


Figure 2.6: A schematic representation of the distributed controller for a 2×2 VSR with 3 voxels, with two sensors in each. Green and magenta arrows represent the connection of the controller with inputs (sensors) and output (actuator), respectively. We remark that—in this representation—we disregard inputs that should come from a certain direction if no neighbor is present.

Each dynamical system takes as input a vector

$$\boldsymbol{x}_i^{(k)} \in \mathbb{R}^{|\boldsymbol{r}_i|+4n_c} = \left[\boldsymbol{r}_i^{(k)}\ \boldsymbol{i}_{i,\uparrow}^{(k)}\ \boldsymbol{i}_{i,\rightarrow}^{(k)}\ \boldsymbol{i}_{i,\downarrow}^{(k)}\ \boldsymbol{i}_{i,\leftarrow}^{(k)}\right], \tag{2.8}$$

where $\boldsymbol{r}_i^{(k)} \in \mathbb{R}^{|\boldsymbol{r}_i|}$ are the local sensor readings, and the remaining terms are the input communication values coming from the adjacent voxel placed above, right, below, left—if the voxel is not connected to another voxel on a given side, the corresponding vector of communication values is the zero vector $\mathbf{0} \in \mathbb{R}^{n_c}$. Each dynamical system is governed by the following set of equations:

$$\boldsymbol{s}_i^{(k)} = g_{i,\boldsymbol{\theta}_i}\left(\boldsymbol{x}_i^{(k)}, \boldsymbol{s}_i^{(k-1)}\right), \tag{2.9}$$

$$\boldsymbol{y}_i^{(k)} = \left[a_i^{(k)}\ \boldsymbol{o}_{i,\uparrow}^{(k)}\ \boldsymbol{o}_{i,\rightarrow}^{(k)}\ \boldsymbol{o}_{i,\downarrow}^{(k)}\ \boldsymbol{o}_{i,\leftarrow}^{(k)}\right] = f_{i,\boldsymbol{\theta}_i}\left(\boldsymbol{x}_i^{(k)}, \boldsymbol{s}_i^{(k)}\right), \tag{2.10}$$

$$\boldsymbol{s}_i^{(0)} = \boldsymbol{s}_{i,0}, \tag{2.11}$$

where $a_i^{(k)} \in \mathbb{R}$ is the local actuation value, $\boldsymbol{o}_{i,\bullet}^{(k)} \in \mathbb{R}^{n_c}$ are the vectors of output communication values going towards the adjacent voxel placed above, right,

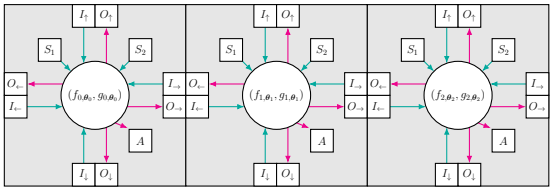


Figure 2.7: A schematic representation of the distributed controller with explicit communication for a 3×1 VSR with two sensors in each voxel and $n_c = 1$ communication channel per side. Green and magenta arrows represent the connection of the controller with inputs (sensors and input communication channels) and outputs (actuator and output communication channels), respectively.

below, left of the voxel, $g_{i,\boldsymbol{\theta}_i} : \mathbb{R}^{|\boldsymbol{r}_i|+4n_c} \times \mathbb{R}^{|\boldsymbol{s}_i|} \to \mathbb{R}^{|\boldsymbol{s}_i|}$ is the local state update function, $f_{i,\boldsymbol{\theta}_i} : \mathbb{R}^{|\boldsymbol{r}_i|+4n_c} \times \mathbb{R}^{|\boldsymbol{s}_i|} \to \mathbb{R}^{1+4n_c}$ is the local output function, and $\boldsymbol{s}_{i,0} \in \mathbb{R}^{|\boldsymbol{s}_i|}$ is the local initial state. All state update and output functions are parameterized in $\boldsymbol{\theta}_i$.

To clarify how communication works, we provide a brief example. Let the voxel $i+1$ be located on the right of voxel i, it holds that $\boldsymbol{o}_{i,\rightarrow}^{(k)} = \boldsymbol{i}_{i+1,\leftarrow}^{(k+1)}$ and $\boldsymbol{o}_{i+1,\leftarrow}^{(k)} = \boldsymbol{i}_{i,\rightarrow}^{(k+1)}$. In other words, communication occurs with one time step of delay among neighbors.

We show a schematic representation of this controller for a 3×1 VSR in Figure 2.7.

Last, we remark that a *non-directional* communication is also possible: in this case the function only outputs n_c communication values, $\boldsymbol{o}_i^{(k)}$, which are equally fed to all Von Neumann neighbors. Thus, Equation (2.10) becomes

$$\boldsymbol{y}_i^{(k)} = \left[a_i^{(k)}\ \boldsymbol{o}_i^{(k)}\right] = f_{i,\boldsymbol{\theta}_i}\left(\boldsymbol{x}_i^{(k)}, \boldsymbol{s}_i^{(k)}\right), \tag{2.12}$$

with $\boldsymbol{o}_{i,\uparrow}^{(k)} = \boldsymbol{o}_{i,\rightarrow}^{(k)}\ \boldsymbol{o}_{i,\downarrow}^{(k)} = \boldsymbol{o}_{i,\leftarrow}^{(k)} = \boldsymbol{o}_i^{(k)} \in \mathbb{R}^{n_c}$, and $f_{i,\boldsymbol{\theta}_i} : \mathbb{R}^{|\boldsymbol{r}_i|+4n_c} \times \mathbb{R}^{|\boldsymbol{s}_i|} \to \mathbb{R}^{1+n_c}$.

2.2 Artificial Neural Networks (for Robot Control)

ANNs are computational models inspired by biological neural networks. They consist of artificial neurons interconnected by weighted connections that mirror the varying intensities of synaptic contacts. Neurons are modeled as electrically excitable nerve cells that transmit electrical signals through synapses [274].

ANNs aim to replicate the behavior and adaptive features of biological neural circuits. Consequently, they are often employed to tackle complex problems, such as modeling intricate relationships between inputs and outputs [161]. In

this manuscript, we (mostly) utilize ANNs as robotic controllers, aiming to infer the relationship between a robot state and perceptions, and its actions.

Since we use ANNs to control agents that operate over time, we inherently introduce the concept of time within the ANNs. Specifically, we simulate the evolution of the ANN over time in a discrete manner, with a time resolution of Δt_h. Notably, Δt_h does not necessarily need to align with the time resolution of the robot physical simulation, allowing the ANN to be updated at different instants compared to the robot simulation.

Due to the stateful nature of some neural models employed, the ANNs can be regarded as dynamical systems, where the outputs depend not only on the current inputs but also on the previous history of the system, reflected by the *state* of the ANN. Thus, an ANN can be described using the following notation:

$$\boldsymbol{s}^{(h)} = g\left(\boldsymbol{x}^{(h)}, \boldsymbol{s}^{(h-1)}\right), \tag{2.13}$$

$$\boldsymbol{y}^{(h)} = f\left(\boldsymbol{x}^{(h)}, \boldsymbol{s}^{(h)}\right), \tag{2.14}$$

$$\boldsymbol{s}^{(0)} = \boldsymbol{s}_0, \tag{2.15}$$

where $\boldsymbol{s}^{(h)} \in \mathbb{R}^{|\boldsymbol{s}|}$ is the state of the ANN at time step h, $\boldsymbol{x}^{(h)} \in \mathbb{R}^m$ is the input at h, $\boldsymbol{y}^{(h)} \in \mathbb{R}^n$ is the output at h, $g : \mathbb{R}^m \times \mathbb{R}^{|\boldsymbol{s}|} \to \mathbb{R}^{|\boldsymbol{s}|}$ is the function determining the state update given an input, $f : \mathbb{R}^m \times \mathbb{R}^{|\boldsymbol{s}|} \to \mathbb{R}^n$ is the function determining the output given the current state and input, and $\boldsymbol{s}_0$ is the initial state.

Since the very first computational model proposed by McCulloch and Pitts [255] in 1943—the *perceptron*—a wide variety of ANN models have emerged. These models differ in various aspects, such as neuron modeling and network architecture, and cover a range of features including simplicity, computational efficiency, and biological resemblance. In this manuscript, we consider a subset of models that capture different aspects, as we will discuss in the following sections.

Specifically, we examine Multi-Layer Perceptrons (MLPs), Recurrent Neural Networks (RNNs), and Spiking Neural Networks (SNNs). While the first two models have been extensively used for robot control over the past decades [344], SNNs have only recently started being applied in this domain. We focus on these three models, among the many that are applicable to the task of robot control [174], because we believe they represent different trade-offs between parameter size and state size. The former serves as a proxy for information processing expressiveness, i.e., the diversity of processing that the ANN can perform, while the latter serves as a proxy for memory size. Additionally, the parameter size directly affects the search space of the optimization process: generally, the more parameters an ANN has, the longer it takes to find suitable values. Finally, we note that more complex ANN models, such as Long Short Term Memory (LSTM) networks [244], are often used in controllers with large input spaces,

such as those resulting from vision sensors. In contrast, the robots in our scenario are usually equipped with simple sensors that generate low-dimensional inputs, as we shall see in the following chapters of this work.

2.2.1 Multilayer Perceptrons

The first model we consider is the MLP, which we leverage for the control of MSRs in Chapters 3 to 7, 11 and 12, and for the control of non-modular rigid robots in Chapters 9 and 10.

The MLP is a fully connected feed forward ANN, where each neuron is a perceptron [255, 362] and neurons are organized in layers. The MLP has no state, which means that, in terms of Equations (2.13) to (2.15), $|\boldsymbol{s}|= 0$, g, and $\boldsymbol{s}_0$ are undefined, and f takes only $\boldsymbol{x}^{(h)}$ as argument.

The processing of $\boldsymbol{y}^{(h)} = f\left(\boldsymbol{x}^{(h)}\right)$ in an MLP is based on the processing occurring in a single neuron:

$$v_{l,i}^{(h)} = \varphi^{\mathrm{P}}\left(\left(\sum_{j=1}^{j=m_{l-1}} w_{l,i,j} v_{l-1,j}^{(h)}\right) - b_{l,i}\right), \tag{2.16}$$

where $\varphi^{\mathrm{P}} : \mathbb{R} \to \mathbb{R}$ is the *activation function* of the perceptron, $v_{l,i}$ is the activation level of the i-th neuron in the l-th layer, $w_{l,i,j}$ is the synaptic weight associated with the synapse connecting the i-th neuron in the l-th layer with the j-th neuron in the $(l-1)$-th layer, $b_{l,i}$ is the bias of the i-th neuron in the l-th layer, and m_l is the size, i.e., the number of neurons in the l-th layer. Since MLPs are organized into layers, respectively one input layer, zero or more hidden layers, and one output layer, the computation described by Equation (2.16) is performed layer by layer, meaning that the outputs of the previous layer become the inputs of the following one, for each time step h. Equation (2.16) can be written more concisely by replacing the summation with vector product:

$$v_{l,i}^{(h)} = \varphi^{\mathrm{P}}\left(\boldsymbol{w}_{l,i}^{\mathsf{T}} \boldsymbol{v}_{l-1}^{(h)} - b_{l,i}\right). \tag{2.17}$$

Hence, for the l-th layer:

$$\begin{aligned} \boldsymbol{v}_l^{(h)} &= \begin{bmatrix} \varphi^{\mathrm{P}}\left(\boldsymbol{w}_{l,1}^{\mathsf{T}} \boldsymbol{v}_{l-1}^{(h)} - b_{l,1}\right) \\ \vdots \\ \varphi^{\mathrm{P}}\left(\boldsymbol{w}_{l,i}^{\mathsf{T}} \boldsymbol{v}_{l-1}^{(h)} - b_{l,m_l}\right) \end{bmatrix}^{\mathsf{T}} \\ &= \boldsymbol{\varphi}_{\boldsymbol{\theta}_l}^{\mathrm{LP}}\left(\boldsymbol{v}_{l-1}^{(h)}\right), \end{aligned} \tag{2.18}$$

where $\boldsymbol{\theta}_l \in \mathbb{R}^{(m_{l-1}+1)m_l}$ is a numerical vector containing all the synaptic weights and biases and parameterizing the function $\boldsymbol{\varphi}^{\text{LP}}_{\boldsymbol{\theta}_l} : \mathbb{R}^{m_{l-1}} \rightarrow \mathbb{R}^{m_l}$ that describes the processing for one layer of neurons. Finally, the entire MLP processing can be completely described in terms of Equation (2.14) by:

$$\begin{aligned}\boldsymbol{y}^{(h)} = \boldsymbol{v}^{(h)}_{l^*+1} &= \boldsymbol{\varphi}^{\text{LP}}_{\boldsymbol{\theta}_{l^*+1}}\left(\boldsymbol{\varphi}^{\text{LP}}_{\boldsymbol{\theta}_{l^*}}\left(\cdots\left(\boldsymbol{\varphi}^{\text{LP}}_{\boldsymbol{\theta}_1}\left(\boldsymbol{v}^{(h)}_0\right)\right)\right)\right) \\ &= f^{\text{MLP}}_{\boldsymbol{\theta}}\left(\boldsymbol{v}^{(h)}_0\right) = f^{\text{MLP}}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(h)}\right), \end{aligned} \tag{2.19}$$

where $\boldsymbol{v}^{(h)}_0 \triangleq \boldsymbol{x}^{(h)}$, l^* is the number of hidden layers in the MLP, with $l = 0$ corresponding to the input layer (for which $m_0 = m$) and $l = l^*+1$ corresponding to the output layer (for which $m_{l^*+1} = n$), and $\boldsymbol{\theta} \in \mathbb{R}^p$ is a numerical vector containing all the parameters of f_{MLP}, with $p = \sum_{l=1}^{l=l^*+1}|\boldsymbol{\theta}_l| = \sum_{l=1}^{l=l^*+1}(m_{l-1}+1)m_l$.

Summarizing, an MLP with l^* hidden layers has a state of size $|\boldsymbol{s}| = 0$ and can be described with a parameter vector of size $|\boldsymbol{\theta}| = \sum_{l=1}^{l=l^*+1}(m_{l-1}+1)m_l$.

Plastic MLPs: Hebbian Learning

Hebbian learning is a type of plasticity that enables an ANN to adapt by modifying its synaptic weights in response to a sequence of inputs. This adaptation is grounded in the concept proposed by Hebb [149], which suggests that the strength of a synapse should vary based on the activation levels of the two neurons it connects. Notably, this form of adaptation operates independently of any measure of the ANN performance (i.e., it does not rely on a reward signal), as it is based solely on the local information available to each synapse, specifically the activation of the pre-synaptic and post-synaptic neurons. Consequently, Hebbian learning can be regarded as a form of *unsupervised learning*.

In this manuscript, we leverage Hebbian learning in Chapter 5, where we use it to investigate the effects of brain plasticity in the artificial domain. Specifically, we use a form of Hebbian learning called the ABCD model (named after the four coefficients defining the update rule) [49] applied to an MLP.

At each time step h, the weight $w_{l,i,j}$ of a synapse connecting the j-th neuron of the $(l-1)$-th layer with the i-th neuron of the l-th layer is updated as follows:

$$w^{(h)}_{l,i,j} = w^{(h-1)}_{l,i,j} + \eta\left(A_{l,i,j}v^{(h-1)}_{l-1,j} + B_{l,i,j}v^{(h-1)}_{l,i} + C_{l,i,j}v^{(h-1)}_{l-1,j}v^{(h-1)}_{l,i} + D_{l,i,j}\right), \tag{2.20}$$

where η is the learning rate, $v^{(h-1)}_{l,i}$ is the post-synaptic activation value at the previous time step, and $v^{(h-1)}_{l-1,j}$ is the pre-synaptic value at the previous time step. The role of the learning rate η is to make the adaptation of the synaptic weights faster or slower: the larger its value, the faster the adaptation. The $A_{l,i,j}$, $B_{l,i,j}$,

$C_{l,i,j}$, and $D_{l,i,j}$ coefficients determine how each synaptic weight updates over time, starting from a $w^0_{l,i,j}$; these coefficients can hence be optimized to achieve a desired behavior. As the notation suggests, these coefficients are, in principle, different for each synapse [109, 463], although other configurations are also possible [331, 112], e.g., a single network-wise set of A, B, C, D parameters. Similarly, also η could be fixed network-wise (as in this case), optimized network-wise, or even fixed/optimized synapse-wise.

Differently from the non-plastic MLP, the Hebbian MLP (H-MLP) is a stateful ANN: the state resides in its current weights, $\boldsymbol{s}^{(h)} = \boldsymbol{w}^{(h)}$, and the state update function $g^{\text{H-MLP}}_{\boldsymbol{\theta}}$ is the function responsible for updating them, reported in Equation (2.20). The output function $f^{\text{H-MLP}}_{\boldsymbol{\theta}}$ coincides with that of the MLP, $f^{\text{MLP}}_{\boldsymbol{\theta}}$. An H-MLP with l^* hidden layers and non-parameterized weights can be described with a vector of size $|\boldsymbol{\theta}| = 4\sum_{l=1}^{l=l^*+1}(m_{l-1}+1)m_l$, the 4 resulting from the four A, B, C, D coefficients. In the case where the initial weights are also parameterized, the number of parameters becomes $|\boldsymbol{\theta}| = 5\sum_{l=1}^{l=l^*+1}(m_{l-1}+1)m_l$.

2.2.2 Recurrent Neural Networks

The second neural model that we consider are RNNs with perceptrons as elementary units, which include memory. In this case, as for MLPs, the output of each unit is computed with φ^{P} and neurons are organized in layers, but the way neurons are connected is different. In particular, the topology we opt for consists of an input layer and an output layer, which are built similarly to those of MLPs, and l^* *recurrent layers* in between them. The input synapses fully connect neurons of the input layer with those of the first recurrent layer, while the output synapses fully connect the last recurrent layer with the output layer. In each recurrent layer, instead, every neuron is connected with every neuron in the previous layer and in the same layer (also considering auto-synapses). In this case, to avoid infinite recursion, the inputs of the recurrent neurons at time h are the outputs of the same layer at the previous time step $h-1$, together with those computed by the input layer at time h.

In mathematical terms this becomes, for the single neuron in the l-th recurrent layer:

$$v^{(h)}_{l,i} = \varphi^{\text{P}}\left(\boldsymbol{w}^{\intercal}_{l,i}\boldsymbol{v}^{(h)}_{l-1} + \boldsymbol{w}'^{\intercal}_{l,i}\boldsymbol{v}^{(h-1)}_{l} - b_{l,i}\right), \tag{2.21}$$

where $\boldsymbol{w}'_{l,i} \in \mathbb{R}^{m_l^2}$ is the vector of the synaptic weights for the recurrent synapses, i.e., those connecting the recurrent layer neurons with each other. More concisely,

for the entire recurrent l-th layer:

$$\begin{aligned}\boldsymbol{v}_l^{(h)} &= \begin{bmatrix} \varphi^{\mathrm{P}}\left(\boldsymbol{w}_{l,1}^{\intercal}\boldsymbol{v}_{l-1}^{(h)} + \boldsymbol{w}_{l,1}^{\prime,\intercal}\boldsymbol{v}_l^{(h-1)} - b_{l,1}\right) \\ \vdots \\ \varphi^{\mathrm{P}}\left(\boldsymbol{w}_{l,m_l}^{\intercal}\boldsymbol{v}_{l-1}^{(h)} + \boldsymbol{w}_{l,m_l}^{\prime,\intercal}\boldsymbol{v}_l^{(h-1)} - b_{l,m_l}\right) \end{bmatrix}^{\intercal} \\ &= \boldsymbol{\varphi}_{\boldsymbol{\theta}_l}^{\mathrm{RP}}\left(\boldsymbol{v}_{l-1}^{(h)}, \boldsymbol{v}_l^{(h-1)}\right), \end{aligned} \tag{2.22}$$

with $\boldsymbol{\theta}_l \in \mathbb{R}^{(m_{l-1}+1)m_l+m_l^2}$ and $\boldsymbol{\varphi}_{\boldsymbol{\theta}_l}^{\mathrm{RP}} : \mathbb{R}^{m_{l-1}} \times \mathbb{R}^{m_l} \to \mathbb{R}^{m_l}$.

In terms of the Equations (2.13) and (2.14), and by associating the state $\boldsymbol{s} = \begin{bmatrix} \boldsymbol{v}_1^{(h)} & \dots & \boldsymbol{v}_{l^*}^{(h)} \end{bmatrix} \in \mathbb{R}^{\sum_l m_l}$ with the activation level of the recurrent layers, we can write:

$$\boldsymbol{s}^{(h)} = g_{\boldsymbol{\theta}}^{\mathrm{RNN}}\left(\boldsymbol{v}_0^{(h)}, \boldsymbol{s}^{(h-1)}\right) = g_{\boldsymbol{\theta}}^{\mathrm{RNN}}\left(\boldsymbol{x}^{(h)}, \boldsymbol{s}^{(h-1)}\right), \tag{2.23}$$

$$\boldsymbol{y}^{(h)} = \boldsymbol{\varphi}_{\boldsymbol{\theta}_{l^*+1}}^{\mathrm{RP}}\left(\boldsymbol{v}_{l^*}^{(h)}\right) = f_{\boldsymbol{\theta}}^{\mathrm{RNN}}\left(\boldsymbol{s}^{(h)}\right), \tag{2.24}$$

with $\boldsymbol{v}_0^{(h)} \triangleq \boldsymbol{x}^{(h)}$.

From these equations the statefulness of RNNs becomes glaring, as opposed to the stateless MLPs. In fact, here the computation at time step h directly and explicitly depends from the results obtained at $h-1$. This is a form of memory of the ANN, which can keep track of previous events and neural activity, possibly re-using the information stored for future computations.

Summarizing, an RNN has a state with size $|\boldsymbol{s}| = \sum_{l=1}^{l=l^*} m_l$ and can be described with a parameter vector with size $|\boldsymbol{\theta}| = \sum_{l=1}^{l=l^*+1} |\boldsymbol{\theta}_l| = (m_{l^*}+1)m_{l^*+1} + \sum_{l=1}^{l=l^*}(m_{l-1}+1)m_l + m_l^2$.

We employ RNNs in comparison with other neural architectures in Chapter 3. However, we remark that the distributed controller with explicit communication for MSRs has itself a form of recurrence, thus also Chapters 4 to 7 and 11 encompass a loose form of RNN.

2.2.3 Spiking Neural Networks

The two aforementioned neural models, MLPs and RNNs, are both simple and fairly computationally efficient, yet they lack in terms of biological plausibility. Hence, we move towards SNNs for our third model, where biological resemblance plays a fundamental role [125]. We experimentally compare SNNs against MLPs and RNNs in Chapters 3 and 4 with the goal of understanding if more biologically plausible neural models might offer advantages w.r.t. simpler and less accurate models when employed as robot controllers.

The key element of SNNs is the modeling of the evolution over time of the membrane potential of neurons. The membrane potential can be altered by incoming excitatory or inhibitory neural stimuli, occurring in the form of spikes propagating along synapses. The generation of said spikes, called *firing*, occurs whenever the membrane potential of a neuron exceeds a given threshold. Despite the binary nature of spikes, the intensity of any stimulus received by a neuron is modulated by the strength of the synapse connecting the firing neuron (pre-synaptic neuron) and the neuron receiving the spike (post-synaptic neuron), similarly to what we have seen for MLPs and RNNs in Sections 2.2.1 and 2.2.2.

Hence, the core difference between SNNs and MLPs or RNNs lies in the way information is encoded: in SNNs information is embedded in the distribution of spikes over time, whereas MLPs and RNNs encode information as real values at every instant. Therefore, to use SNNs within a framework designed to be coupled with canonical ANNs, we require additional procedures to convert spike trains to real values and vice versa, to make the SNN consistent with Equations (2.13) and (2.14), where the input is a numerical vector $\boldsymbol{x}^{(h)} \in \mathbb{R}^m$ and the output is a numerical vector $\boldsymbol{y}^{(h)} \in \mathbb{R}^n$. We describe those conversion procedures in Section 2.2.4.

Within the SNN paradigm, several neuron models exist [170], which all share the main concepts derived from neuroscience. In this manuscript, we opt for the Leaky Integrate and Fire (LIF) model, simulated in discrete time, which offers a reasonable trade-off between computational efficiency and biological resemblance. The LIF model represents the neural membrane as a capacitor, the potential of which can be increased or decreased by inputs (excitatory or inhibitory), and exponentially decays with time.

Concerning the architecture of the SNN, we keep a fully connected feed forward layered topology, similarly to an MLP, where each neuron is an LIF neuron rather than a perceptron. At each neural simulation time step h, the membrane potential $v_{l,i}^{(h)}$ of the i-th LIF neuron of the l-th layer is:

$$\begin{aligned} v_{l,i}^{(h)} &= v_{l,i}^{(h-1)} - \Delta t_h \lambda_v v_{l,i}^{(h-1)} + \sum_{j=1}^{j=m_{l-1}} w_{l,i,j} \nu_{l-1,j}^{(h)} \\ &= v_{l,i}^{(h-1)}(1 - \Delta t_h \lambda_v) + \boldsymbol{w}_{l,i} \boldsymbol{\nu}_{l-1}^{(h)} \\ &= \varphi_{\boldsymbol{w}_{l,i}}^{\mathsf{M}} \left(\boldsymbol{\nu}_{l-1}^{(h)}, v_{l,i}^{(h-1)} \right), \end{aligned} \tag{2.25}$$

with $\nu_{l-1,j}^{(h)} \in \{0, 1\}$ carrying the pre-synaptic neuron spike. After the update, if the membrane potential $v_{l,i}^{(h)}$ exceeds a threshold $\vartheta_{l,i}^{(h)}$, the neuron outputs a spike, i.e., $\nu_{l,i}^{(h)} = 1$, and the membrane potential is reset to its resting value v_{rest},

otherwise $\nu_{l,i}^{(h)} = 0$. Formally:

$$v_{l,i}^{(h)} = \begin{cases} \varphi_{\boldsymbol{w}_{l,i}}^{\text{M}}\left(\boldsymbol{\nu}_{l-1}^{(h)}, v_{l,i}^{(h-1)}\right) & \text{if } \varphi_{\boldsymbol{w}_{l,i}}^{\text{M}}\left(\boldsymbol{\nu}_{l-1}^{(h)}, v_{l,i}^{(h-1)}\right) \leq \vartheta_{l,i}^{(h)} \\ v_{\text{rest}} & \text{otherwise} \end{cases}$$
$$= \varphi_{\boldsymbol{w}_{l,i}}^{v\text{LIF}}\left(\boldsymbol{\nu}_{l-1}^{(h)}, \vartheta_{l,i}^{(h)}, v_{l,i}^{(h-1)}\right), \tag{2.26}$$

$$\nu_{l,i}^{(h)} = \begin{cases} 0 & \text{if } \varphi_{\boldsymbol{w}_{l,i}}^{\text{M}}\left(\boldsymbol{\nu}_{l-1}^{(h)}, v_{l,i}^{(h-1)}\right) \leq \vartheta_{l,i}^{(h)} \\ 1 & \text{otherwise} \end{cases}$$
$$= \varphi_{\boldsymbol{w}_{l,i}}^{\nu\text{LIF}}\left(\boldsymbol{\nu}_{l-1}^{(h)}, \vartheta_{l,i}^{(h)}, v_{l,i}^{(h-1)}\right), \tag{2.27}$$

which can be ported to the analogous layer-wise form, with $\boldsymbol{\varphi}_{\boldsymbol{\theta}_l}^{v\text{LIF}} : \{0,1\}^{m_{l-1}} \times \mathbb{R}^{m_l} \times \mathbb{R}^{m_l} \to \mathbb{R}^{m_l}$ and $\boldsymbol{\varphi}_{\boldsymbol{\theta}_l}^{\nu\text{LIF}} : \{0,1\}^{m_{l-1}} \times \mathbb{R}^{m_l} \times \mathbb{R}^{m_l} \to \{0,1\}^{m_l}$, where $\boldsymbol{\theta}_l = [\boldsymbol{w}_{l,1} \ \dots \ \boldsymbol{w}_{l,m_l}] \in \mathbb{R}^{m_{l-1}m_l}$ is, similarly to the MLP, a numerical vector containing all the synaptic weights (with no biases in this case) for the l-th layer.

Plastic SNNs: Homeostasis

To enhance the LIF model, we also introduce the biological concept of *homeostatic plasticity*. Homeostasis is a self regulatory mechanism occurring at various sites of living organisms, which aims at re-establishing equilibrium in contrast to strong stimuli that could unbalance a system [427]. In our case, homeostasis operates as a firing rate regulator, acting on the threshold $\vartheta_{l,i}^{(h)}$ of neurons, to prevent excessive or too scarce activity:

$$\vartheta_{l,i}^{(h)} = \min\left(\vartheta_{l,i}^{(h-1)}, \sum_{j=1}^{j=m_l} w_{l,i,j}\right) + \psi_{l,i}^{(h)}$$
$$= \varphi_{\boldsymbol{w}_{l,i}}^{\vartheta\text{LIF}}\left(\psi_{l,i}^{(h)}, \vartheta_{l,i}^{(h-1)}\right), \tag{2.28}$$

with $\psi_{l,i}^{(h)}$ being a parameter updated as:

$$\psi_i^{(h)} = \begin{cases} \psi_{l,i}^{(h-1)} + \psi_{\text{inc}} & \text{if } \nu_{l,i}^{(h-1)} = 1 \\ \psi_{l,i}^{(h-1)} - \psi_{l,i}^{(h-1)} \lambda_\psi \Delta t_h & \text{otherwise.} \end{cases}$$
$$= \varphi^{\psi\text{LIF}}\left(\nu_{l,i}^{(h-1)}, \psi_{l,i}^{(h-1)}\right). \tag{2.29}$$

Both the equations describing homeostasis can be ported to the corresponding layer-wise form, where $\boldsymbol{\varphi}_{\boldsymbol{\theta}_l}^{\vartheta\text{LIF}} : \mathbb{R}^{m_l} \times \mathbb{R}^{m_l} \to \mathbb{R}^{m_l}$ and $\boldsymbol{\varphi}^{\psi\text{LIF}} : \{0,1\}^{m_l} \times \mathbb{R}^{m_l} \to \mathbb{R}^{m_l}$.

Overall, the processing of the l-th layer of an SNN is described by:

$$\boldsymbol{\psi}_l^{(h)} = \boldsymbol{\varphi}^{\psi\text{LIF}}\left(\boldsymbol{\nu}_l^{(h-1)}, \boldsymbol{\psi}_l^{(h-1)}\right), \tag{2.30}$$

$$\boldsymbol{\vartheta}_l^{(h)} = \boldsymbol{\varphi}_{\boldsymbol{\theta}_l}^{\vartheta\text{LIF}}\left(\boldsymbol{\psi}_l^{(h)}, \boldsymbol{\vartheta}_l^{(h-1)}\right), \tag{2.31}$$

$$\boldsymbol{v}_l^{(h)} = \boldsymbol{\varphi}_{\boldsymbol{\theta}_l}^{v\text{LIF}}\left(\boldsymbol{\nu}_{l-1}^{(h)}, \boldsymbol{\vartheta}_l^{(h)}, \boldsymbol{v}_l^{(h-1)}\right), \tag{2.32}$$

$$\boldsymbol{\nu}_l^{(h)} = \boldsymbol{\varphi}_{\boldsymbol{\theta}_l}^{\nu\text{LIF}}\left(\boldsymbol{\nu}_{l-1}^{(h)}, \boldsymbol{\vartheta}_l^{(h)}, \boldsymbol{v}_l^{(h-1)}\right). \tag{2.33}$$

It can be seen that the state, for the l-layer, is given by $\boldsymbol{s}_l = [\boldsymbol{v}_l\ \boldsymbol{\vartheta}_l\ \boldsymbol{\psi}_l] \in \mathbb{R}^{3m_l}$, for which we can write:

$$\begin{aligned}\boldsymbol{s}_l^{(h)} &= \begin{bmatrix} \boldsymbol{\varphi}_{\boldsymbol{\theta}_l}^{v\text{LIF}}\left(\boldsymbol{\nu}_{l-1}^{(h)}, \boldsymbol{\vartheta}_l^{(h)}, \boldsymbol{v}_l^{(h-1)}\right) \\ \boldsymbol{\varphi}_{\boldsymbol{\theta}_l}^{\vartheta\text{LIF}}\left(\boldsymbol{\psi}_l^{(h)}, \boldsymbol{\vartheta}_l^{(h-1)}\right) \\ \boldsymbol{\varphi}^{\psi\text{LIF}}\left(\boldsymbol{\nu}_l^{(h-1)}, \boldsymbol{\psi}_l^{(h-1)}\right) \end{bmatrix}^{\intercal} \\ &= \boldsymbol{\varphi}_{\boldsymbol{\theta}_l}^{s\text{LIF}}\left(\boldsymbol{\nu}_{l-1}^{(h)}, \boldsymbol{s}^{(h-1)}\right). \end{aligned} \tag{2.34}$$

The processing of the entire SNN with l^* hidden layers may be described, in terms of Equations (2.13) and (2.14), as:

$$\begin{aligned}\boldsymbol{s}^{(h)} &= g_{\boldsymbol{\theta}}^{\text{SNN}}\left(\boldsymbol{\nu}_{-1}^{(h)}, \boldsymbol{s}^{(h-1)}\right) \\ &= g_{\boldsymbol{\theta}}^{\text{SNN}}\left(\boldsymbol{x}^{(h)}, \boldsymbol{s}^{(h-1)}\right), \end{aligned} \tag{2.35}$$

$$\boldsymbol{y}^{(h)} = \boldsymbol{\nu}_{l^*+1}^{(h)} = f_{\boldsymbol{\theta}}^{\text{SNN}}\left(\boldsymbol{s}^{(h)}\right), \tag{2.36}$$

with $\boldsymbol{\nu}_{-1}^{(h)} \triangleq \boldsymbol{x}^{(h)}$.

Summarizing, an SNN with Homeostasis (SNN-H) with l^* hidden layers, has a state with size $|\boldsymbol{s}| = \sum_{l=0}^{l=l^*+1} |\boldsymbol{s}_l| = 3\sum_{l=0}^{l=l^*+1} m_l$ and can be described with a parameter vector with size $|\boldsymbol{\theta}| = \sum_{l=1}^{l=l^*+1} |\boldsymbol{\theta}_l| = \sum_{l=1}^{l=l^*+1} m_{l-1}m_l$. Moreover, the SNN-H processing depends on the following network-wise parameters: Δt_h, λ_v, v_{rest}, λ_ψ, and ψ_{inc}.

We also employ an SNN without homeostasis, that we denote with just SNN in the following. For this neural model, each $\vartheta_{l,i}^{(h)}$ is the same and remains constant, i.e., $\vartheta_{l,i}^{(h)} = \vartheta_0, \forall l, i, h$; hence, the state of the SNN for each l-th layer does not include $\boldsymbol{\vartheta}_l$ and $\boldsymbol{\psi}_l$. As a consequence, an SNN with l^* hidden layers and without homeostasis has a state with size $|\boldsymbol{s}| = \sum_{l=0}^{l=l^*+1} |\boldsymbol{s}_l| = \sum_{l=0}^{l=l^*+1} m_l$ and can be described with a parameter vector with the same size of the SNN-H, i.e., $|\boldsymbol{\theta}| = \sum_{l=1}^{l=l^*+1} m_{l-1}m_l$. Finally, the SNN processing depends on the following network-wise parameters: Δt_h, λ_v, v_{rest}, and ϑ_0.

2.2.4 Embedding Different ANN Models in Robot Controllers

Given the differences between the considered neural models, highlighted in Sections 2.2.1 to 2.2.3, it logically follows that not all ANNs can be embedded in a robot controller alike. In particular, a key role is played by the different ways in which information is stored and processed: MLPs and RNNs rely on real values at each neural simulation time step h, whereas SNNs encode information in spike trains and in their distribution over time. Therefore, we distinguish between the former and the latter models when used as robot controllers.

MLPs and RNNs as Robot Controllers

Using these ANNs as controllers is straightforward, as they process real values just like the robot control system, which takes the sensor readings (and potentially communication values) as inputs and outputs the control values for the actuators. Therefore, it is sufficient to query those ANNs at each control time step k to compute the desired values. Hence, in this case, the neural simulation frequency coincides with the chosen control frequency f_c.

SNNs as Robot Controllers

As anticipated in Section 2.2.3, unlike canonical ANNs, SNNs do not process real values: at each neural time step h, there is a binary value, i.e., either there is a spike or not, which does not contain meaningful information itself. Instead, information is stored within the distribution of spikes over time, hence we need to convert real values to a spike distribution, i.e., to a spike train, and vice versa, in order to embed SNNs in a standard robot controller.

Here, we take inspiration from the most common conversion method, that is *rate coding* [42], where real values are mapped to frequencies, which are then used to generate spike trains [452], and vice versa. Moreover, it is interesting to note that rate coding contributes substantially to the homeostatic mechanism, which operates as a firing frequency regulator. We remark, though, that other types of coding exist, which naturally couple with other types of regulation mechanisms, such as temporal coding, which enhances the benefits of Spike-Timing-Dependent Plasticity (STDP) [226] for local learning [137].

As a consequence of the encoding choice, the neural simulation frequency in SNNs, $f_h = \frac{1}{\Delta t_h}$, needs to be larger than the control frequency f_c—i.e., the control interval has to be longer than the neural simulation interval. In Chapters 3 and 4, we set $f_h = 1\,\mathrm{kHz}$, corresponding to a time resolution of $1\,\mathrm{ms}$, which is a commonly used value in the literature [170].

Depending on the value of f_c we use two rate coding flavors. The first is the standard version of rate coding. Given a sensor reading $r^{(k)}$ we convert it to a

spike train as follows. We first compute $f^{(k)}$ by linearly scaling $r^{(k)}$ between f_{min} and f_{max}:

$$f^{(k)} = r^{(k)}(f_{\text{max}} - f_{\text{min}}) + f_{\text{min}}. \tag{2.37}$$

Then we generate a spike train with spike frequency $f^{(k)}$, i.e., we emit a spike every $\left\lfloor \frac{f_h}{f^{(k)}} \right\rfloor$ neural time steps—note that this could also be done according to some probabilistic distribution [17]. Formally speaking, we assign each value $\nu^{(h)}$ in the spike train for $h = k \left\lfloor \frac{f_h}{f_c} \right\rfloor, \ldots, (k+1) \left\lfloor \frac{f_h}{f_c} \right\rfloor - 1$, i.e., for the k-th control interval, as:

$$\nu^{(h)} = \begin{cases} 1 & \text{if } \exists n \in \mathbb{N} \text{ s.t. } h = h_0 + n \left\lfloor \frac{f_h}{f^{(k)}} \right\rfloor \\ 0 & \text{otherwise.} \end{cases} \tag{2.38}$$

where we set $h_0 = k \left\lfloor \frac{f_h}{f_c} \right\rfloor$ to the starting point of the control interval.

Concerning the output conversion, we compute the average frequency of the output spike train in the control interval, the inverse of the average inter-spike timing, and we scale it considering the maximum frequency f_{max} to compute $a^{(k)}$:

$$a^{(k)} = \frac{f_h}{f_{\text{max}}} \sum_{h=k\left\lfloor \frac{f_h}{f_c} \right\rfloor}^{h=(k+1)\left\lfloor \frac{f_h}{f_c} \right\rfloor - 1} \nu^{(h)}. \tag{2.39}$$

In this way, we obtain a value of $a^{(k)}$ defined in $[0, 1]$ which we can rescale according to the actual domain of definition of $a^{(k)}$, e.g., $[-1, 1]$. We set $f_{\text{min}} = 5\,\text{Hz}$ and $f_{\text{max}} = 50\,\text{Hz}$ for biological plausibility. Thus, we can rely on this coding when $f_c \ll f_{\text{max}} = 50\,\text{Hz}$.

With higher control frequencies, i.e., when $f_c \approx f_{\text{max}}$, we modify the standard rate coding as follows. For the input conversion, we employ Equation (2.38), but we set h_0 to the neural time step of the last spike occurred before this control interval. For the output conversion we modify Equation (2.39) to consider the last n_w control intervals, i.e., from $k - n_w - 1$ to k, as in a moving average. Hence, the output conversion equation becomes

$$a^{(k)} = \frac{1}{n_w} \frac{f_h}{f_{\text{max}}} \sum_{h=(k-n_w+1)\left\lfloor \frac{f_h}{f_c} \right\rfloor}^{h=(k+1)\left\lfloor \frac{f_h}{f_c} \right\rfloor - 1} \nu^{(h)}. \tag{2.40}$$

We remark that the value-to-spikes and the spikes-to-value conversions are stateful themselves. In particular, when $f_c \approx f_{\text{max}}$, an unbounded number of previous control time steps may impact on the value-to-spikes conversion (since the last previous spike might have occurred in any previous control time step). In the spikes-to-value conversion, exactly n_w previous control time steps are considered.

2.3 Evolutionary Optimization (of Robots)

An Evolutionary Algorithm (EA) [100, 84] is a meta-heuristic, population-based optimization algorithm. EAs are inspired by Darwin's theory of evolution [78], where individuals in a population compete for limited resources. This environmental pressure leads to natural selection, where the fittest individuals have a higher likelihood of survival and reproduction.

To solve an optimization problem, a *fitness function* must be defined to evaluate the performance of individuals. Similar to natural evolution, individuals with higher fitness are more likely to survive and reproduce, leading to an expected overall increase in the population's fitness.

In EAs, an individual represents a candidate solution (*phenotype*) for the problem, which can be internally represented (*genotype*) either as the solution itself or as a specific data structure. The mapping between genotype and phenotype is commonly known as *encoding*. The genotype-phenotype distinction not only enhances the biological analogy of EAs, but also facilitates the reuse of techniques developed for evolving certain genotypes, requiring users to only customize the encoding for the problem at hand.

The general structure of an EA is illustrated in Figure 2.8. At first, the population is initialized, either randomly or based on prior knowledge about the problem. Next, a subset of the population is chosen for offspring generation, a process known as *parent selection*, which is analogous to mating in biology: through *recombination* or *crossover*, the offspring inherits traits from parents. Additionally, *mutation* may be applied to the new individual to increase diversity within the population, promoting the exploration of new areas in the solution space. In some cases, new individuals can be generated solely through mutation. After the new set of individuals is created, fitness is assessed for all individuals to simulate environmental pressure. Based on these fitness evaluations, decisions are made about which individuals will proceed to the next generation and which will be eliminated. This entire cycle repeats until a termination condition is satisfied, such as achieving a specified fitness level or exceeding a computational budget, e.g., a set number of generations.

2.3.1 Evolutionary Robotics

Turning our focus back to robotics, the actions of a robot are driven towards the achievement of a specific goal, known as a *task*. It is usually necessary to optimize a robot for a given task, and EAs are particularly effective and efficient at exploring the robot design space, often requiring only the definition of a suitable fitness measure. While defining this fitness measure can be challenging [90], it is generally easier than manually designing a solution.

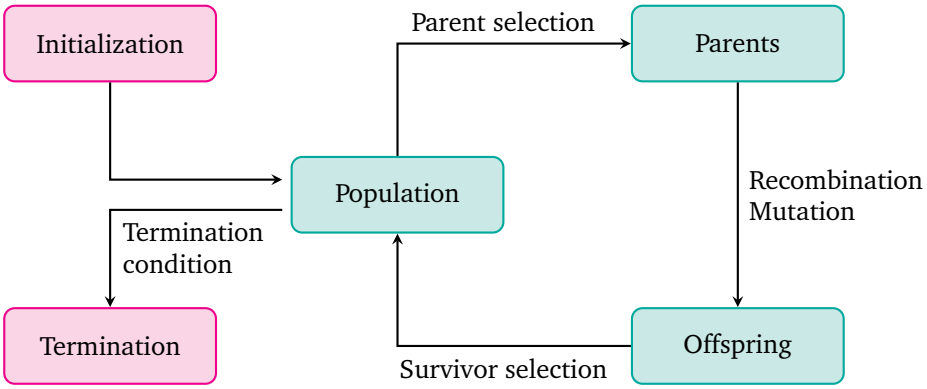


Figure 2.8: General scheme of an EA [100].

The application of EC to robotics is termed Evolutionary Robotics (ER) [94, 318], which involves optimizing the robot (or its parts) for a given task using EC. Robotics has several peculiarities which make EC a relevant optimization technique:

1. Optimization can target different robot features, such as the controller or the body. Although most examples of ER are either aimed at optimizing the controller, the body of the agent, or both, there are also some more peculiar examples, such as the optimization of the robot sensory apparatus [107]. Even when only the controller is optimized, the body plays a crucial role in interacting with the environment through morphological computation [147], as described by the embodied cognition paradigm [334].

2. Robots are not static artifacts, i.e., they change over time, from simple spatial configuration changes to more complex scenarios like growth [204], autotomy [81], or malfunctions [250, 206]. This allows interaction between evolutionary and life time-scales, enabling adaptation at both levels.

3. Real-world application of EC steps (evaluation, selection, and variation) can be costly, unscalable, or dangerous. Hence, optimization is often performed in simulation, leveraging fast computation and the parallelizable nature of EAs. However, simulations rarely capture every subtle aspect of reality, leading to the reality gap problem [430, 372], where the optimized robot behaves differently in real-world conditions compared to simulations.

2.3.2 Relevant Examples of EAs

In the following we mention and describe some relevant examples of EAs employed in the manuscript.

Genetic Algorithms

Genetic Algorithms (GAs) [158] are the most "standard" form of EAs. We detail the pseudo-code of a GA in Algorithm 2.1.

A GA iteratively evolves a fixed size population P of n_{pop} solutions. First, solutions are generated with the `init()` function, which can be customized according to the chosen representation. Then, at each iteration, the GA builds an offspring P' of n_{off} solutions by selecting parent individuals with a tournament selection (with n_{tour} individuals) and then applying a variation operator. Such operator is implemented by the `variate(...)` function: it can take the form of crossover, mutation, or a combination of the two, according to the use case. The GA then extracts an elite of size n_{elite} from the parent population. After that, the GA merges the parents and offspring and applies truncation selection to retain only the best $n_{\text{pop}} - n_{\text{elite}}$ solutions, which will be merged with the elite to form the population at the next iteration. The GA stops iterating when at least n_{evals} solutions have been evaluated. At the end of evolution, the GA returns the best solution in the final population.

Evolutionary Strategies

Evolutionary Strategies (ES) arise from real-valued function optimization and hence employ vectors of real numbers as genotypes [84]. We frequently utilize ES in this manuscript to optimize the weights of ANNs, effectively performing a form of neuroevolution. This optimization approach has gained recognition as a scalable alternative to Reinforcement Learning (RL) [371], garnering significant attention across various communities.

We report the pseudo-code of an ES in Algorithm 2.2. At first, n_{pop} individuals, i.e., numerical vectors $\boldsymbol{\theta}$, are put in the initially empty population, all generated by assigning to each element of the vector of size p a value sampled from the uniform distribution $U([-1, 1])$. Subsequently, the algorithm proceeds in iterations, limited by a budget specified in number of evaluations n_{evals}. At every iteration, an elite of the population is chosen to generate $n_{\text{pop}} - 1$ children, each obtained by adding values sampled from a normal distribution $N(0, \sigma)$ to each element of the element-wise mean $\boldsymbol{\mu}$ of all parents. The generated offspring, together with the fittest individual of the previous generation, end up forming the population of the next generation, which maintains the fixed size n_{pop}.

```
1  function evolve():
       // initialization
2      P ← ∅
3      while |P| < n_pop do
4          s ← init()
5          P ← P ∪ {s}
6      end
       // generations
7      n ← n_pop
8      while n < n_evals do
           // build offspring
9          P' ← ∅
10         while |P'| < n_off do
11             n ← n + 1
12             s_1 ← tournament(P, n_tour, q)
13             s_2 ← tournament(P, n_tour, q)
14             s ← variate(s_1, s_2)
15             P' ← P' ∪ {s}
16         end
           // extract elite from parents
17         P'' ← P
18         while |P''| > n_elite do
19             P'' ← P'' \ {worst(P'', q)}
20         end
           // merge parents and offspring
21         P ← P ∪ P'
           // apply truncation selection
22         while |P| > n_pop − n_elite do
23             P ← P \ {worst(P, q)}
24         end
           // merge selected population with elite
25         P ← P ∪ P''
26     end
27     return best(P, q)
28 end
```

Algorithm 2.1: The general scheme of a GA. The functions $\texttt{best}(P, q)$ and $\texttt{worst}(P, q)$ return the best or worst solution in P according to the fitness function q.

```
1  function evolve():
       // initialization
2      P ← ∅
3      foreach i ∈ {1, ..., n_pop} do
4          P ← P ∪ {0 + U([−1, 1])^p}
5      end
       // generations
6      n ← n_pop
7      while n < n_evals do
           // select parents elite
8          P′ ← P
9          while |P′| > n_elite do
10             P′ ← P′ \ {worst(P′, q)}
11         end
12         μ ← mean(P′)
13         P″ ← {best(P, q)}
14         while |P″| < n_pop do
15             P″ ← P″ ∪ {μ + N(0, σ)^p}
16             n ← n + 1
17         end
18         P ← P″
19     end
20     return best(P, q)
21 end
```

Algorithm 2.2: The scheme of a simple ES. The functions `best`(P, q) and `worst`(P, q) return the best or worst solution in P according to the fitness function q.

Genetic Programming

Genetic Programming (GP) extends the principles of GAs to the domain of computer programs, enabling the automatic generation and optimization of programs to solve specific tasks [200]. GP can utilize various representations for the programs it evolves: we describe the ones employed in the manuscript below. Regardless of the representation, GP usually follows the same evolutionary loop of a GA (see Algorithm 2.1) with suitable implementations of the `init()` and `variate()` functions, although combining GP with other EAs, e.g., multi-objective EAs, is also possible.

The main advantage offered by GP over other representations, such as ANNs optimized for controlling robotic agents or performing regression, lies in the compactness and expressivity of the solutions GP generates. Even without explicitly constraining the size of the solution, GP produces results composed of simple user-defined building blocks that are interconnected, thereby improving their interpretability. In this manuscript, we adopt GP in Chapters 8 to 12 for this reason.

Tree-based Genetic Programming. Tree-based GP represents programs as tree structures, where each node corresponds to a function belonging to a set H, and leaf nodes represent variables or constants [201, 246]. Function nodes perform operations and have children nodes as arguments, while terminal nodes provide input values or constants. The output of the program is obtained at the root by recursively evaluating the functions until leaf nodes are met.

In tree-based GP the genotype corresponds to the tree itself, thus initialization and genetic operators act directly on the tree. For initialization the most common strategy is *ramped half-and-half* [246], that generates each tree as follows. First, it samples a target tree depth $d \in [d_{\text{min}}, d_{\text{max}}]$. Then, it selects either the *grow* or *full* algorithm with probability $1/2$. Both algorithms recursively pick random nodes for the new tree starting from the root; the difference between the two algorithms is that grow samples between both functional nodes and terminal nodes, whereas full only samples functional nodes. When they reach the target depth, both algorithms only sample terminal nodes.

Concerning the genetic operators, the most common ones are *subtree crossover* and *subtree mutation*. Subtree crossover consists in replacing a random subtree of one parent with a randomly chosen subtree of the other parent: both subtrees are picked to ensure the child tree has a maximum depth of d_{max}. In subtree mutation a random subtree is replaced with a newly generated tree, ensuring a maximum depth of d_{max} for the child.

We opt for tree-based GP in Chapters 8 and 11. In the former, we choose it due to its demonstrated effectiveness in addressing Symbolic Regression (SR) problems [211] and its potential for interpretability, which allows us to explore the subjective nature of this concept. In the latter, we select it for similar reasons,

aiming to experiment with compact and potentially interpretable controllers for MSRs.

Cartesian Genetic Programming. Cartesian Genetic Programming (CGP) represents programs as graphs resulting from connections among nodes placed in grid in a Cartesian plane [276, 275]. Each node represents a function, which can use either the outputs of the nodes of the previous layers or the program inputs as arguments. The final outputs of the program are collected from the outputs of n_{out} selected nodes.

Without loss of generality, we can consider a uni-dimensional grid, i.e., a sequence of nodes of length n_{nodes}. Hence, to fully determine a CGP graph, we need (1) n_{out} indexes to specify the program outputs, and (2) n_{nodes} tuples $(h, i_1, \ldots, i_{m_{\text{ar}}})$, specifying the function and its arguments for each node, where m_{ar} is the maximum arity of the functions available in the function set H, 2 in our case. We remark that for each of the tuples, h is an index bounded by the cardinality of H, whereas each input index can range from 0 to $n_{\text{in}} + j - 1$, where j indicates the current node position. Thus, the genome of a CGP graph is a sequence of bounded integers of size $n_{\text{out}} + (1 + m_{\text{ar}})n_{\text{nodes}}$.

To initialize each genome, we sample a uniform distribution over the allowed integer values for each position. As variation operator, we use *int-flip mutation*, i.e., we sample a new integer value in the allowed interval, with a different probability for node inputs p_i, node functions p_f, and program outputs p_o.

We utilize CGP in Chapters 9, 10 and 12 due to its strength in representing multivariate functions and leveraging sub-structure modularity. Furthermore, its fixed-length representation enhances computational efficiency, allowing us to exploit GPU acceleration. We employ CGP to develop interpretable controllers without sacrificing performance, addressing different types of robots: rigid robots in Chapters 9 and 10 and MSRs in Chapter 12.

Linear Genetic Programming. In Linear Genetic Programming (LGP), programs are lists of instructions from a programming language, where the result of each instruction is assigned to a register from a predefined set [44]. The inputs of the program are copied into the first n_{in} registers, whereas the outputs are taken from the last n_{out} registers. If we consider the information flow in a LGP program, we can interpret it as a Directed Acyclic Graph (DAG), and hence LGP falls under the category of Graph-based Genetic Programming (GGP) [117].

To describe a LGP program, each instruction is determined by specifying (1) the index of the register to be assigned, (2) the function to execute from those available in the function set H, and (3) the sequence of arguments to be passed to the function $(i_1, \ldots, i_{m_{\text{ar}}})$, expressed in terms of register indexes ($m_{\text{ar}} = 2$ being the maximum arity of functions in H). Hence, to fully determine

a program of n_{lines} lines, the genome is a sequence of bounded integers of size $(2 + m_{\text{ar}})n_{\text{lines}}$. In this case the bounds for register indexes are constituted by the amount of available registers n_{reg}, as even those not explicitly assigned before are initialized to 0 and are available to select.

To generate each genome, we sample a uniform distribution over the allowed integer values for each position. Regarding the genetic operators, we apply *one-point crossover* (constraining instructions, i.e., program lines, to be atomic) followed by *int-flip mutation*, with a different probability for the left-hand side of each program line, i.e., the registers to be assigned, p_a, functions p_f, and function inputs p_i.

We employ LGP in Chapter 9 as an alternative to CGP, choosing it for similar reasons.

NSGA-II: a Multi-Objective EA

Non-dominated Sorting Genetic Algorithm II (NSGA-II) [86] is a multi-objective EA that seeks to find a diverse set of solutions that represent the Pareto front in the objective space. We report the pseudo-code of NSGA-II in Algorithm 2.3.

Initially, n_{pop} solutions are generated with the `init()` function and stored in the population P. Successively, the first generation of offspring is computed, selecting parents with tournament selection based on Pareto dominance, and applying a suitable variation operator upon them. Then, the core loop of the algorithm begins, with survival selection occurring first, followed by a new offspring generation.

Survival selection, detailed in Algorithm 2.4, proceeds iteratively, by subsequently promoting to the next population the Pareto fronts. This proceeds until the considered Pareto front would make the next population exceed the target size of n_{pop}. In that case, the solutions to be promoted are selected according the crowded-comparison operator $\prec_n$. Such operator sorts solutions according to Pareto dominance first, and, in case of ties, resorts to the *crowding distance* for comparing solutions. In simple terms, the crowding distance measures the normalized inter-solution distance in the space defined by the fitness functions $\boldsymbol{q}$ (note that it is also possible to compute it w.r.t. the parameter space [85]). The sorting is performed to promote solutions with higher distance to others, i.e., in less crowded areas, to implement a sort of diversity preservation mechanism.

After individuals are selected for survival, NSGA-II computes a new generation, selecting parents with tournament selection and applying variation. As for survival selection, the tournament selection relies on the crowded-comparison operator $\prec_n$, aiming at promoting fit solutions while maintaining a good coverage over the different objectives.

At the end of the algorithm, all solutions belonging to the first Pareto front are returned.

```
1  function evolve():
       // initialization
2      P ← ∅
3      while |P|< n_pop do
4          s ← init()
5          P ← P ∪ {s}
6      end
       // first generation
7      while |P'|< n_pop do
           // tournament selection based on Pareto dominance
8          s_1 ← tournament(P, n_tour, q)
9          s_2 ← tournament(P, n_tour, q)
10         s ← variate(s_1, s_2)
11         P' ← P' ∪ {s}
12     end
13     P ← P ∪ P'
14     n ← 2n_pop
       // generations
15     while n < n_evals do
           // survival selection
16         P ← survivalSelection(P, n_pop, q)
           // build offspring
17         P' ← ∅
18         while |P'|< n_pop do
19             n ← n + 1
               // tournament selection based on crowded-comparison operator ≺_n
20             s_1 ← tournament(P, n_tour, ≺_n)
21             s_2 ← tournament(P, n_tour, ≺_n)
22             s ← variate(s_1, s_2)
23             P' ← P' ∪ {s}
24         end
           // merge parents and offspring
25         P ← P ∪ P'
26     end
27     return nonDominated(P, q)
28 end
```

Algorithm 2.3: The scheme of NSGA-II. The function `survivalSelection`$(P, n_{\text{pop}}, \boldsymbol{q})$ is the core point of the algorithm; it is detailed in Algorithm 2.4. The function `nonDominated`$(P, \boldsymbol{q})$ extracts the first Pareto front in the population P according to the fitness functions $\boldsymbol{q}$.

```
1  function survivalSelection(P, n_pop, q):
2      P' ← ∅
3      while |P'| < n_pop do
           // select Pareto front
4          F ← nonDominated(P, q)
           // add Pareto front if it fits entirely in the population
5          if |P'| + |F| ≤ n_pop then
6              P' ← P' ∪ F
7              P ← P \ F
8          else
               // select individuals according to crowded-comparison operator ≺_n
9              while |P'| < n_pop do
10                 s ← best(F, ≺_n)
11                 P' ← P' ∪ {s}
12             end
13         end
14     end
15     return P'
16 end
```

Algorithm 2.4: The survival selection mechanism employed in NSGA-II (see Algorithm 2.3). The function `best`$(P, \prec_n)$ returns the best solution in P according to the fitness function $\prec_n$, whereas the function `nonDominated`$(P, \boldsymbol{q})$ returns the first Pareto front in the population P according to the fitness functions $\boldsymbol{q}$.

MapElites: a Quality-Diversity EA

Quality-Diversity (QD) algorithms are algorithms that search for a large collection of both *diverse* and high-performing solutions. The role of this collection is to cover the range of possible solution types as much as possible, and to contain the best solution for each type [69].

MAP-Elites (ME) is a QD algorithm that creates a map of high-performing solutions at each point in a space defined by user-specified dimensions of diversification [291]. We show the pseudo-code of the default version of ME in Algorithm 2.5.

```
1  function evolve():
       // initialize the empty map of elites (archive)
2      χ ← ∅
3      n ← 0
4      while n < n_evals do
5          if n < n_pop then
               // generate first individuals with init function
6              s ← init()
7          else
               // generate new individuals from parents
8              s′ ← sample(χ)
9              s ← variate(s′)
10         end
           // extract descriptors of solution
11         d ← descriptorFunction(s)
           // if the archive has no element for the given descriptors or the
              element is worse than the generated one (assuming maximization of
              the fitness), store the new one
12         if χ(d) = ∅ or q(χ(d)) < q(s) then
13             χ(d) ← s
14         n ← n + 1
15     end
16     return χ
17 end
```

Algorithm 2.5: The scheme of ME. The function `descriptorFunction(`s`)` computes the descriptors of the solution s.

ME works by storing solutions in an *archive* χ, also known as *repertoire* or *map*. At first n_{pop} solutions are randomly generated, whereas successively new solutions are generated by applying variation to solutions sampled from the archive. The decision if a solution s is stored in the archive is based on two

elements: the fitness of the solution $q(s)$ and its descriptors $\boldsymbol{d}$, computed by the `descriptorFunction(`s`)`. Namely, if there is no element in the archive corresponding to the same descriptors $\boldsymbol{d}$, the solution is stored. Conversely, if there is already an element, the decision is based on local competition, i.e., the best solution among the existing one and the new one is kept.

The descriptors of the solution are user-specified and should capture the *features* of the solution that are considered relevant by the user. Clearly, if the space of descriptors is real valued, a discretization is required. The standard ME version relies on a simple grid discretization, whereas CVT-ME [434, 435] discretizes the space of descriptors into an archive according to the Centroidal Voronoi Tessellation (CVT), providing a more uniform and efficient coverage of the descriptors space, regardless of its dimensionality.

In this manuscript, specifically in Chapters 10 and 12, we consistently use CVT-ME. We employ ME as a tool to diversify solutions based on various criteria, ultimately investigating whether this partially bio-inspired mechanism can enhance the optimization of robot controllers that are both interpretable and effective.

I

Biological Insights as a Foundation for Autonomous Embodied Artificial Intelligence

In this part of the thesis, we study how to integrate biological principles into the domain of Artificial Intelligence (AI) for robotics. More specifically, we investigate whether bio-mimicry can enhance the performance and autonomy of embodied AI agents, while also examining if the findings observed in-silico can reflect and illuminate phenomena in the biological domain.

Our analysis begins with a comparison of how different models of Artificial Neural Networks (ANNs) perform as controllers for embodied agents in Chapters 3 and 4. We evaluate various ANN models with differing levels of simplicity and biological fidelity. Next, we investigate the biological phenomenon of plasticity in artificial systems, focusing on brain plasticity in Chapter 5 and morphological plasticity in Chapter 6. We compare plastic agents with non-plastic ones, highlighting potential advantages of this paradigm and other notable differences. Finally, we simulate synaptic pruning in ANNs in Chapter 7, examining the effects of reducing over-parametrization in the ANNs employed to control embodied agents.

3

Comparison of Evolved Neural Network Models

A key question in robotics consists in determining the most suitable neural controller for a specific robot morphology and sensory setup in a given environment, especially when it comes to robots with non trivial dynamic behaviors as Voxel-based Soft Robots (VSRs). It has been observed that ANNs with state might handle the dynamic characteristics of VSR bodies and their morphological computation effectively. In this chapter, we evaluate four types of ANNs as controllers: Multi-Layer Perceptrons (MLPs), Recurrent Neural Networks (RNNs), and Spiking Neural Networks (SNNs) with and without homeostasis. We test three robot morphologies for locomotion, each evaluated in simulation with three different types and numbers of sensors. We optimize neural network controllers through Neuroevolution (NE), and assess the results based on effectiveness, efficiency, and generalization. We also systematically analyze the robots behaviors. Our findings indicate that RNNs generally perform better, while MLPs often prove to be the least effective, especially for robots with fewer sensors. On the other hand, SNNs excel in generalization, and homeostasis tends to be advantageous. Lastly, we note that RNNs and SNNs with homeostasis tend to produce a broader range of behaviors.

This chapter is based on the following publication: **G. Nadizar**, E. Medvet, S. Nichele, and S. Pontes-Filho. An experimental comparison of evolved neural network models for controlling simulated modular soft robots. *Applied Soft Computing*, page 110610, 2023 [299].

3.1 Introduction

ANNs have been extensively and effectively employed for a diverse array of tasks, from language interactions [380] and playing strategic board games [102] to generating real-time robot commands [227]. Of particular relevance to problems where temporal dynamics are crucial, RNNs stand out as a notable class of ANNs with state. Having a state means these networks possess memory capability, allowing outputs from a neuron to influence future computations of the same node through cycles. Consequently, RNNs can be viewed as dynamical systems, which makes them potentially well-suited as controllers for embodied agents that operate in real-time, responding to their environment and previous actions.

However, in the field of modular robotics, there is insufficient understanding regarding the advantages of using recurrent neural controllers versus other types of neural controllers for various morphologies and tasks.

To address this research gap, in this chapter we examine four variants of ANN—three of which include a state—and conduct a comprehensive comparison of their performance as controllers for VSRs. We note that VSRs are particularly relevant in this context because their bodies consist of interconnected soft modules, allowing them to display a wide range of dynamic behaviors [410].

These dynamic processes, enabled and conducted by the robot bodies, are commonly referred to as morphological computations [294]. To align with the body dynamics and fully leverage their potential, we explore various evolved neural network models—specifically, different ANN models optimized with Evolutionary Computation (EC). We focus on simple, stateless MLPs, RNNs, and SNNs, with and without homeostasis. These four variants differ in both parameter size and state size; the former indicating the information processing capability of the network, while the latter representing its memory capacity. Homeostasis, a biological mechanism that maintains system stability amid abrupt changes in input stimuli, is implemented in SNNs as homeostatic plasticity, which regulates the firing rate of a neuron by adjusting the threshold of the membrane potential [427].

Based on the hypothesis that recurrent and spiking neural models could be advantageous for VSRs, we carry out an extensive experimental campaign in simulation. This involves nine robots with varying shapes (biped, worm, and "comb", a form of multiped) and sensory setups (few or many, different types such as proximity and touch), all optimized through NE. We assess the evolved ANNs for their performance in the locomotion task, which involves moving as far as possible from the initial position. We evaluate the networks using metrics that include search effectiveness (measured by the velocity of the evolved robots), search efficiency (based on the evolutionary time required to achieve the best robot), and generalization ability (assessed by the robot velocity under unknown conditions). Additionally, we conduct a systematic analysis of the robots behav-

iors using various metrics and visualize the results through Principal Component Analysis (PCA) for dimensionality reduction. Our findings confirm that ANNs with state are particularly well-suited for controlling VSRs.

We find that:

1. RNNs are the most effective models while MLPs are on average the weakest network controllers for robots with few sensors.

2. SNNs appear to be more capable of generalization than other types of controllers. One interesting observation is that the mechanism of homeostasis is beneficial across a wide variety of scenarios, indeed providing an additional level of fast adaptation during the lifetime of the robotic agents.

3. ANNs with state such as RNNs and SNNs are able to produce a wide repertoire of diverse behaviors, which may be argued to be a beneficial property for robots to be deployed in unknown or changing environments [338].

3.2 Related Works

Controlling a robot to complete a specific task is a complex challenge that can be approached in various ways, from traditional control theory to more innovative AI methods. A common strategy is to equip the robot with a "brain", often in the form of an ANN [344, 28], which handles the computation needed to drive the actions of the robot. In contrast, the concept of morphological computation suggests that the robot "body" itself can perform the computation, making the brain secondary or even unnecessary [294]. The embodied cognition paradigm [334] attempts to reconcile these opposing views by localizing intelligence—i.e., the source of computation—in the interaction between body and brain for both real and artificial agents.

Clearly, this paradigm does not apply equally across the entire robotics field, as some robot morphologies are inherently better suited for hosting computation than others. Intuitively, a more dynamic body, as seen in soft robots [87], tends to have a greater share of computational power residing within the body itself [264]. VSRs naturally fit this paradigm and therefore serve as an excellent test bed for evaluating the body-brain interaction in a controlled simulated environment.

A similar argument applies to robotic controllers: a particular type of brain, or ANN, may or may not be suitable for a specific robot or task. Among the diverse ANN models, one of the key factors in determining their appropriateness is the degree to which they support computation in harmony with the body dynamics. In other words, the choice of ANN is often influenced by features such as having a state (memory) [121], the ability to perform self-regulation and/or

unsupervised learning [305, 331], or being adapted to specific types of inputs (e.g., high-dimensional visual data) [148]. While controllers with memory may be better suited for complex tasks requiring deep environmental cognition, the complex dynamics introduced by such controllers may interfere with the robot body dynamics, which motivates our study. Conversely, learning and adaptation are often desirable, especially when the controller is part of a broader optimization process [99], though the added complexity required for auto-adaptation can sometimes detract from the robot effectiveness. Thus, selecting an appropriate neural model is not straightforward, necessitating a comparison of different options. Jin et al. [174] reviewed various ANNs for a robot manipulation task, including many models featured in our study, as well as Echo State Networks, Dual Networks, and Central Pattern Generators. Similar to their work, we aim to compare different ANN models for a single task. However, unlike the cited study, we focus on simulated modular robots performing locomotion and conduct an experimental evaluation rather than a survey.

The MLP [255, 362] is the most widely used ANN model, commonly chosen for various robot types and applications. MLPs have been successfully applied to VSRs as both centralized and distributed controllers, leveraging the modularity of these robots [338]. Notably, Ferigo et al. [109] incorporated unsupervised Hebbian learning into MLPs within VSRs to boost their performance and generalization capabilities (we will conduct a similar study in Chapter 5).

RNNs are another frequently used ANN model in robot control [417, 216]. While they share the simplicity of MLPs, RNNs include a state, making them well-suited for tasks that require memory, such as navigation [4, 483]. Although RNNs have not been directly applied to VSRs, the distributed controller with explicit communication [260] (presented in Section 2.1.3) intrinsically utilizes recurrence in the information processing within the robot.

Unlike MLPs and RNNs, SNNs have only recently started to gain traction in robotics, though pioneering work on using them for controlling modular robots dates back to 2003 [311]. SNNs were originally introduced to mimic biological processes [125, 170], but are now valued for their unique computational features. They enable homeostasis [427]—a form of parametric unsupervised learning—and are highly energy-efficient, especially when paired with neuromorphic hardware [392]. Bing et al. [33] provide an overview of SNN applications in robotics. Notably, the work by Spaeth et al. [392, 393] is highly relevant to this study. They employ SNNs as Central Pattern Generators (CPGs) [168, 403] to control the contraction and expansion of simulated robot muscles, similar to how voxels function in VSRs. However, the CPGs in these studies are manually designed with inhibitory cycles among neuron groups to create rhythmic muscle activation [193]. SNNs are also capable of creating cognitive maps in spatial or temporal dimensions, serving as a form of spatial memory. This feature enables

them to address tasks beyond locomotion, such as robot path planning and state predictions [40, 240, 472]. Moreover, SNNs can facilitate associative learning, where robots learn associations between objects and actions [37, 239, 223].

To make each of these ANNs effective as robot controllers, an optimization or training phase is essential. Given the variety of models, we use NE for optimization, as it is applicable to all the ANNs considered in this study. Unlike Reinforcement Learning (RL), NE does not require domain-specific knowledge to design rewards (as seen in [32]) and can therefore be applied to a wide range of tasks. Additionally, NE does not require differentiability. For a comprehensive overview of NE-based methods for optimizing ANN-based controllers, especially for navigation tasks, we refer the reader to [172].

3.3 Neuroevolution of VSR Neural Controllers

In this chapter we employ a centralized controller for VSRs, which is thoroughly described in Section 2.1.3. Such controller is a dynamical system defined by two parametric functions $f_{\boldsymbol{\theta}}$ and $g_{\boldsymbol{\theta}}$. These functions determine the way in which the VSR reacts to different conditions, i.e., its behavior. Naturally, the goal of optimization is to find the most effective parameters $\boldsymbol{\theta}$ to induce a behavior which leads the VSR to successfully accomplish a given task.

Since we are employing ANNs as controllers and evolutionary techniques for optimization, the process of optimizing the controller parameters $\boldsymbol{\theta}$ can be referred to as *NE*. This means that we are searching the space of ANNs to find the most suitable ones for solving the task at hand. In this study, we keep the neural topology fixed throughout the entire optimization process, unlike other classical neuroevolutionary techniques, as e.g., NeuroEvolution of Augmenting Topologies (NEAT) [401, 400], which also search the space of ANN topologies. Hence, here we consider the space $\mathbb{R}^p$ of ANN numerical parameters $\boldsymbol{\theta}$ as search space, and aim to the find the optimum therein. Although one could, in principle, include all possible ANN parameters in the search space, thus subjecting them to optimization, we limit ourselves to weights and biases (only weights for SNNs), keeping all other free parameters fixed to hand-set values (i.e., the activation function φ^P and SNN parameters Δt_h, λ_h, v_{rest}, λ_ψ, and ψ_{inc}).

Therefore, we reduce the search of an effective VSR neural controller to a numerical optimization problem, which we tackle with the simple variant of Evolutionary Strategy (ES) described in Algorithm 2.2. We choose ES as they have not only been recognized as a scalable alternative to RL for continuous control tasks [371], but they have also already proved successful for the NE of VSR controllers [338].

We remark that NE, i.e., the application of an Evolutionary Algorithm (EA) to optimize an ANN (here, just the parameters), is not the only approach for finding

a good controller for a robotic agent performing a given task. In particular, in the last years, RL proved to be a very practical alternative, also in scenarios where a model of the robot-environment systems is not known and where the input and output of the robots are numerical, vectors i.e., in continuous control tasks [355]. However, NE and (deep) RL do differ in a key aspect: while for the former a single measure of quality (the fitness) is needed for capturing the degree of achievement of the task by the robot at the end of one entire attempt (the robot life, or the episode), for RL a measure of achievement (the reward) should continuously be available to the robotic agent at every time step for making the optimization effective. In these terms, the reward can be viewed as a form of supervision (possibly sparse, in practice), which is not needed by agents subjected to NE. Beyond this apparent difference, recent trends are making the boundary between RL and NE more and more fuzzy [371], also for robotic control [405]. Indeed, RL has been used also for VSRs, by Bhatia et al. [32].

3.4 Experimental Evaluation

We perform a thorough experimental evaluation to characterize the peculiarities of various neural models for controlling VSRs. In particular, we investigate along three quantitative axes, which are naturally intertwined: (1) effectiveness, (2) efficiency, and (3) generalization ability. Moreover, we also analyze qualitatively the behavior of the evolved VSRs.

We apply NE to optimize the controller for 9 different simulated robots resulting from the combination of 3 morphologies and 3 sensory apparatuses. For each robot, we experiment with both high and low control frequencies, i.e., $f_c \in \{60\,\text{Hz}, 4\,\text{Hz}\}$, in combination with each one of the four proposed neural models.

We consider the task of locomotion, a common one in Evolutionary Robotics (ER), for which we employ a flat terrain during evolution, and rougher terrains to test the generalization ability of the best individuals obtained.

Finally, we visually inspect and systematically analyze the behavior of each robot, following the pipeline introduced by Pigozzi et al. [338].

3.4.1 Procedure and Parameters

VSRs Morphologies and Sensors

We experiment with three morphologies:

- *biped*  , with 10 voxels enclosed in a 4×3 grid;
- *worm*  , with 10 voxels enclosed in a 5×2 grid;

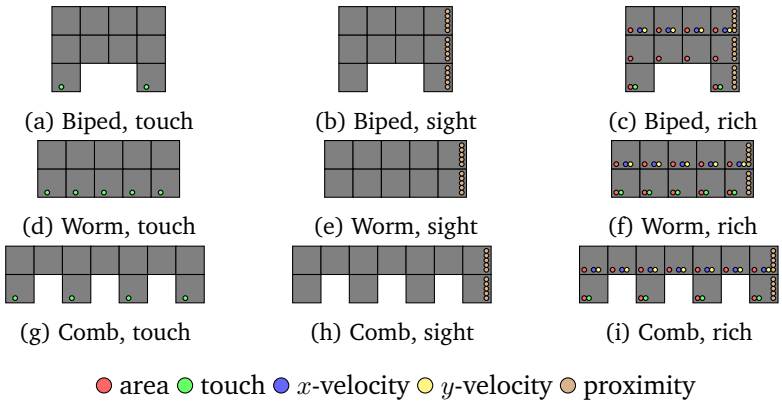


Figure 3.1: VSR morphologies and sensory apparatuses.

- *comb*  , with 11 voxels enclosed in a 7×2 grid.

We equip each morphology with three different sensory apparatuses:

- *touch*, with one touch sensor in each voxel in the bottom row of the grid;
- *sight*, with five proximity sensors in each voxel in the rightmost column of the grid, with α set to $-\frac{\pi}{4}$, $-\frac{\pi}{8}$, 0, $\frac{\pi}{8}$, and $\frac{\pi}{4}\,\mathrm{rad}$ and $\tau_{\text{prox}} = 10\,\mathrm{m}$ (for reference, the side of each voxel is $3\,\mathrm{m}$ long);
- *rich*, with the sensors of touch and sight, one x- and one y-velocity sensor in each voxel in the top row of the grid, and one area sensor in each voxel.

Figure 3.1 shows the 9 VSR bodies resulting from the combination of the 3 morphologies and the 3 sensory apparatuses. Additionally, Table 3.1 summarizes, for each of the 9 VSR bodies, the number n of voxels (ranging from 10 to 11) and the number m of sensors (ranging from 2 to 39).

VSRs Neural Controllers

Concerning the topology of the neural controllers, we employ the same number of hidden layers $l^* = 1$ for all the models and both control frequencies $f_c = 60\,\mathrm{Hz}$ and $f_c = 4\,\mathrm{Hz}$; we set the number of neurons in the single hidden layer to the number of inputs, i.e., $m_1 = m_0 = m$. This results in the number $|\boldsymbol{\theta}|$ of evolvable parameters and the size $|\boldsymbol{s}|$ of the ANN state showed in Table 3.1: $|\boldsymbol{\theta}|$ ranges from 24 for the case biped, touch, SNN to 3521 for the case comb, rich, RNN; $|\boldsymbol{s}|$ ranges from 0, i.e., no state, for all the MLP-based cases to 267 for the case comb, rich, SNN with Homeostasis (SNN-H). We recall that VSRs equipped with SNN

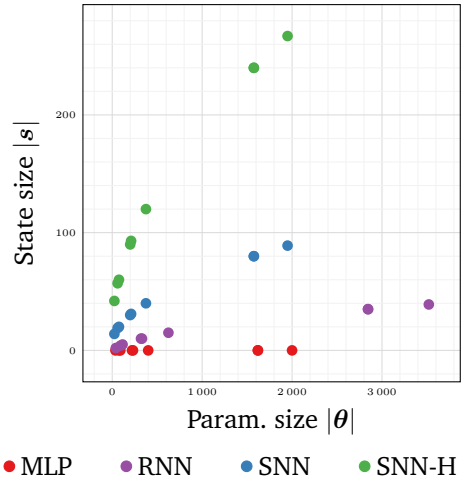


Figure 3.2: Number $|\boldsymbol{\theta}|$ of parameters and state size $|\boldsymbol{s}|$ for each combination of morphology, sensory apparatus, and neural model. The four neural models differ in how $|\boldsymbol{s}|$ and $|\boldsymbol{\theta}|$ depend on the VSR body complexity (i.e., number n of voxels and overall number m of sensors).

and SNN-H neural controllers have some additional state used for the input and output conversion (see Section 2.2.4).

Figure 3.2 summarizes the same data of Table 3.1 in the form of a scatter plot, where each marker corresponds to one case (morphology, sensory apparatus, neural model) and is colored according to the neural model. From this figure, the differences among neural models in terms of $|\boldsymbol{\theta}|$ and $|\boldsymbol{s}|$ are apparent. With the increasing complexity of the VSR body (i.e., with increasing m and n), RNNs are more expressive in terms of computation rather than of memory (i.e., the amount of parameters $|\boldsymbol{\theta}|$ grows faster than the state size $|\boldsymbol{s}|$), while SNNs are more expressive in terms of memory rather than of computation (i.e., $|\boldsymbol{s}|$ grows faster than $|\boldsymbol{\theta}|$).

Concerning the network-wise, non optimizable parameters of the neural models (see Section 2.2), we use $\tanh$ as activation function φ^{P} (in MLP and RNN), $\Delta t_h = 1\,\mathrm{ms}$, $\lambda_v = 0.01/\mathrm{s}$, $v_{\mathrm{rest}} = 0\,\mathrm{mV}$ (in SNN and SNN-H), $\lambda_\psi = 0.01/\mathrm{s}$, $\psi_{\mathrm{inc}} = 0.2\,\mathrm{mV}$ (in SNN-H), and $\vartheta_0 = 1\,\mathrm{mV}$ (in SNN). Finally, in all the experiments, we set the initial state $\boldsymbol{s} = \mathbf{0}$, with the exception of the parts of the state corresponding to the membrane potentials in SNN and SNN-H, that we set, element-wise, to v_{rest} (which is 0 too). For the conversion between real values and spike trains, described in Section 2.2.4, we set $n_w = 5$ after preliminary experimentation.

Morphology	Sens. app.	m	n	Param. size $\lvert\boldsymbol{\theta}\rvert$				State size $\lvert\boldsymbol{s}\rvert$			
				MLP	RNN	SNN	SNN-H	MLP	RNN	SNN	SNN-H
Biped	Touch	2	10	36	40	24	24	0	2	14	42
	Sight	15	10	400	625	375	375	0	15	40	120
	Rich	35	10	1620	2845	1575	1575	0	35	80	240
Worm	Touch	5	10	90	115	75	75	0	5	20	60
	Sight	10	10	220	320	200	200	0	10	30	90
	Rich	35	10	1620	2845	1575	1575	0	35	80	240
Comb	Touch	4	11	75	91	60	60	0	4	19	57
	Sight	10	11	231	331	210	210	0	10	31	93
	Rich	39	11	2000	3521	1950	1950	0	39	89	267

Table 3.1: Number n of voxels and overall number m of sensors for the combinations of morphology and sensory apparatus. For each combination and each neural model, number $|\boldsymbol{\theta}|$ of parameters and state size $|\boldsymbol{s}|$.

Evolutionary Optimization

We use the EA of Algorithm 2.2 with $n_{\text{pop}} = 36$, $n_{\text{evals}} = 20000$, $n_{\text{elite}} = n_{\text{pop}}/4 = 9$, and $\sigma = 0.35$. We set n_{pop} based on the number of cores of the machines where we execute the experiments, in order to fully exploit the inherent capability of this EA to parallelize the fitness computation of the candidate solutions. We set σ based on our experience and on some exploratory experiments.

For each combination of morphology, sensory apparatus, neural model, and control frequency, we execute 10 independent evolutionary runs by varying the random seed. Overall, we execute $3 \cdot 3 \cdot 4 \cdot 2 \cdot 10 = 720$ evolutionary runs.

We optimize VSRs for the task of *locomotion* on a flat terrain. For each candidate solution $\boldsymbol{\theta}$, we initially place a VSR equipped with a neural controller parameterized with $\boldsymbol{\theta}$ on a flat terrain and let the simulation run for $30\,\text{s}$. We hence measure the average x-velocity v_x of the center of mass of the VSR during the last $25\,\text{s}$ of the simulation and use it as the fitness of $\boldsymbol{\theta}$. We discard the first $5\,\text{s}$ of the simulation to let the neural controllers reach a steady gait regime; we also verify that the remaining $25\,\text{s}$ are long enough to allow VSRs to exhibit an assessable behavior.

We limit ourselves to the locomotion task for it is a fundamental and challenging problem in ER, requiring the proper exploitation of the information coming from multiple sensors to effectively guide and coordinate the actuators. Moreover, locomotion is common to many real-world applications, where it can be declined in various nuances, e.g., climbing [32], or it can occur as sub-task, e.g., in exploring unknown environments through path planning [324, 325] or carrying objects around [32].

For the experiments we rely on 2D-VSR-Sim [261], setting each parameter to its default value. We make the software publicly available online[1].

3.4.2 Effectiveness of Evolved Neural Controllers

We hereby consider the effectiveness of the evolved neural controllers, namely, the fitness $v_x^\star$ of the best individual at the end of the evolutionary run.

Figure 3.3 shows $v_x^\star$ for the 72 different cases in the form of a matrix of boxplots. In particular, for each combination of morphology (row of plot), sensory apparatus and control frequency f_c (column of plots), and neural model (box color), the figure considers the 10 values for $v_x^\star$ obtained from the 10 runs with different random seeds. We recall that for $v_x^\star$, the greater, the better.

Regarding the performance of the neural models, Figure 3.3 illustrates that RNNs consistently yield the best results, producing the fastest robots with a significantly higher median $v_x^\star$ compared to other neural models in nearly all cases.

[1] `https://github.com/giorgia-nadizar/NeuralModelsVSR`

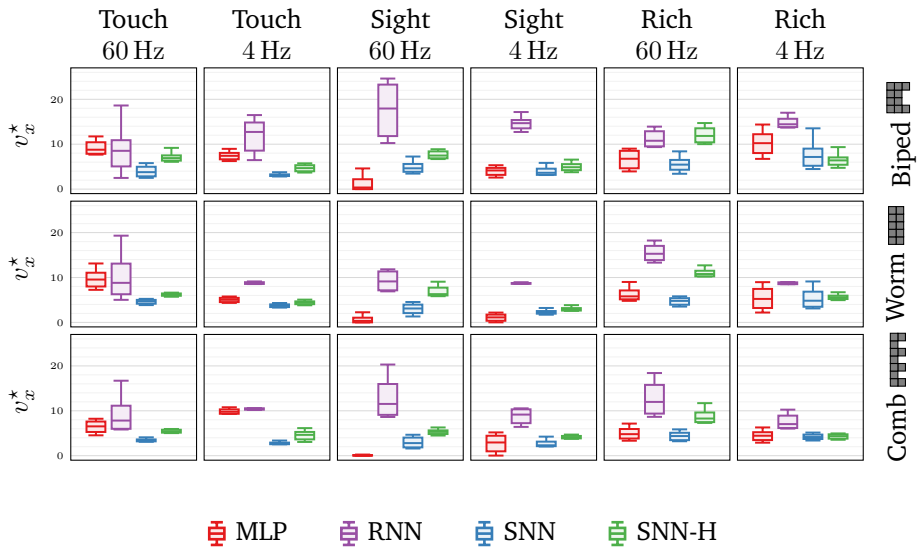


Figure 3.3: Fitness $v_x^\star$ of the best individual at the end of the evolution.

To corroborate this observation, we conduct several statistical significance tests. Specifically, for each combination of morphology, sensory apparatus, and control frequency, we compare pairs of neural models using a one-sided Mann-Whitney U rank test. This test is applied, after verifying the necessary assumptions, with the null hypothesis positing that the distribution of $v_x^\star$ for the first neural model is stochastically lower than or equal to that of the second. Applying a significance level of $\alpha = 0.05$ with Bonferroni correction ($n = 12$), we count the number of combinations where the null hypothesis is rejected for each pair of neural models. The results, presented in Table 3.2, reinforce the findings from Figure 3.3: RNNs outperform the other models in the majority of the 18 cases and are not significantly worse in any scenario.

In addition to revealing the sharp differences in median $v_x^\star$ among the neural models, Figure 3.3 also indicates that RNNs typically exhibit the greatest variability in $v_x^\star$, as evidenced by the vertical extent of the boxes. Despite this variability, RNNs often outperform the other models, even in the least successful evolutionary run.

When narrowing the comparison to the other three models, the experiments presented in Figure 3.3 suggest that there is no definitive superior model. Specifically, MLP and SNN-H outperform each other in a comparable number of cases, though they tend to excel under different conditions (which will be discussed

	MLP	RNN	SNN	SNN-H	Tot.
MLP	–	0	6	5	11
RNN	12	–	17	14	43
SNN	4	0	–	0	4
SNN-H	7	0	13	–	20

Table 3.2: Number of cases (i.e., combinations of morphology, sensory apparatus, and control frequency) for which the neural model on the row is statistically significantly better (in terms of $v_x^\star$, see text) than the neural model on the column. The last column shows the overall number of pairwise comparison for which a neural model is statistically significantly better than one other neural model.

later). Furthermore, both Figure 3.3 and Table 3.2 indicate that homeostasis provides an advantage: in most cases, SNN-H outperforms SNN, while the reverse scenario never occurs.

A second set of insights can be drawn by examining Figure 3.3 with respect to the sensory apparatus used. Specifically, sight appears to be the condition where the differences among neural models are most pronounced when varying morphology and control frequency. Notably, neural models with larger state sizes (i.e., $|\boldsymbol{s}|$) tend to perform better with this sensory apparatus. This observation aligns with previous studies on simulated embodied agents, such as those by Pratt et al. [345], which suggest that long-term planning becomes more critical when coupled with fine-grained sensing, like long-range sight. However, in our scenario of locomotion on flat terrain, long-term planning is likely not a significant factor. Instead, we attribute the superior performance of models with larger memory to the nature of the information provided by the sight apparatus. Sight sensors typically return a constant value of 1, indicating no objects in view, until the robot bends enough for the sensors to detect the terrain, triggering a different value. This means the information from sight is "sparse" over time, requiring the VSR to retain this data to generate an effective gait. Therefore, a larger memory capacity is advantageous. Our interpretation is further supported by two additional observations. First, with sight, MLP consistently performs worse at a control frequency of $f_c = 60\,\mathrm{Hz}$ compared to $f_c = 4\,\mathrm{Hz}$. The lower frequency keeps control values constant for longer, partially compensating for the lack of memory or state of the MLP. Second, with touch sensors, MLP never underperforms relative to SNN or SNN-H. In this case, the sensory input leads to a purely reactive behavior, such as expanding certain voxels upon contact with the ground, where memory is less critical, and the MLP can perform effectively without the need for state.

Lastly, regarding the different morphologies, Figure 3.3 indicates that the

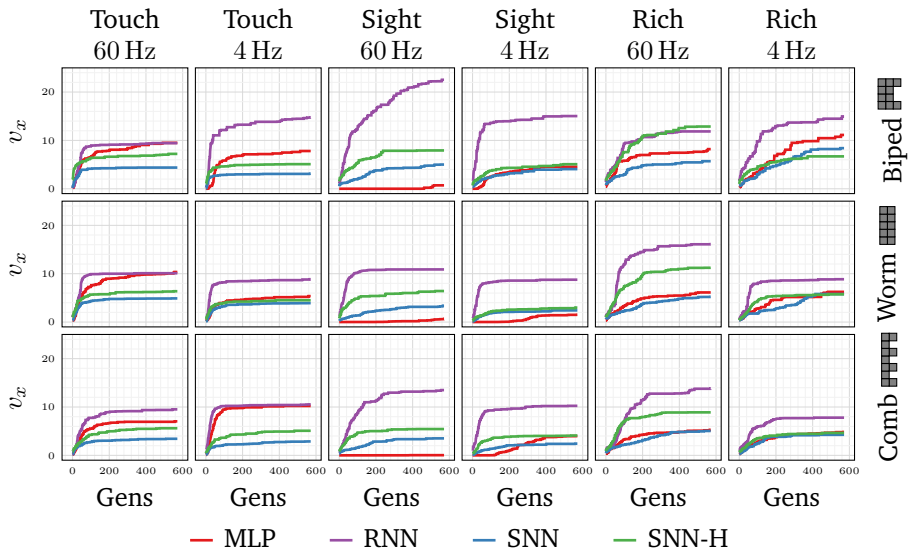


Figure 3.4: Fitness v_x during the evolution.

biped is the most effective morphology for locomotion, a finding that aligns with previous studies such as those by Talamini et al. [410] and Ferigo et al. [109]. This suggests that the biped design inherently supports efficient movement patterns. However, it is important to note that NE successfully discovers controllers that produce effective gaits in almost all cases, with only a few exceptions (MLP with sight and a control frequency of $f_c = 60\,\text{Hz}$ for the worm and the comb). This suggests that while NE is generally robust in optimizing controllers across various morphologies, certain combinations of sensory input and control frequency might challenge specific neural models, particularly in non-bipedal forms like the worm and comb.

3.4.3 Efficiency of Neuroevolution

We consider as search efficiency of NE the effort taken to find an effective solution. Since a definition of effective solution based on the absolute value of the fitness would hardly fit all the considered cases, we consider as effective a solution whose fitness is $\geq 80\,\%$ of the fitness $v_x^\star$ of the best individual at the end of the corresponding run. This way, we define the efficiency as the number g_{80} of generations NE takes to find the first effective solution—the lower the better.

Figure 3.4 shows how the fitness v_x of the best individual varies during the evolution for the 72 cases. For each combination of morphology, sensory ap-

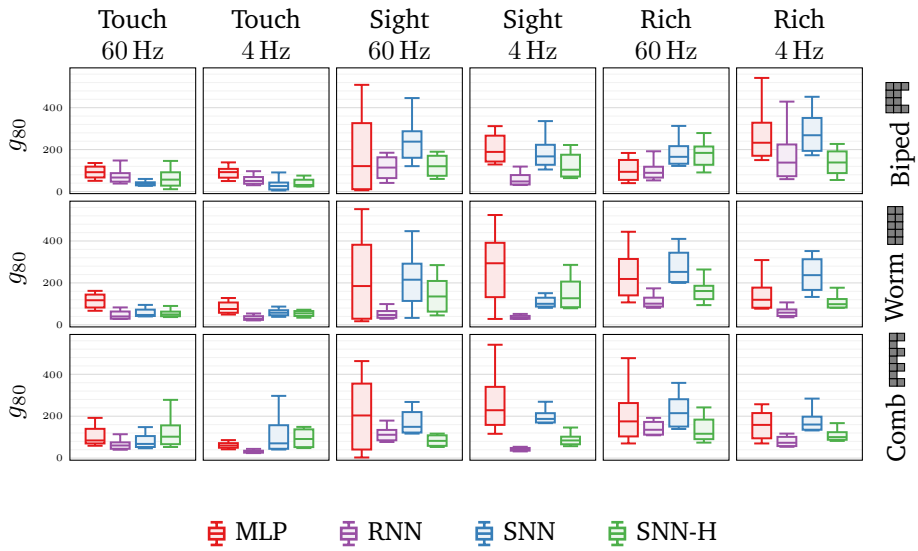


Figure 3.5: Efficiency measured with g_{80}, defined as the number of generations needed to reach a solution with fitness $v_x \geq 0.8v_x^\star$.

paratus, control frequency, and neural model, the figure shows the median v_x across the 10 runs vs. the number of generations, i.e., iterations of the EA of Algorithm 2.2. Based on the data backing Figure 3.4, which confirms the findings drawn in the previous section, we compute g_{80} for each run. Figure 3.5 shows g_{80} for the 72 cases in the form of a matrix of box-plots, organized in the same way as Figure 3.3.

By examining Figure 3.5, it is evident that the differences among neural models in terms of search efficiency are less pronounced compared to the differences in terms of effectiveness. However, the RNN once again stands out as the preferable model, consistently showing lower g_{80} values, indicating a more efficient search process. This observation is further supported by the statistical significance analysis summarized in Table 3.3, where RNN is frequently associated with significantly lower g_{80} values. This means that RNN not only helps in finding the fastest VSR ($v_x^\star$) but also does so in a shorter time frame, showcasing its dual advantage in both search efficiency and effectiveness. This advantage of RNN is particularly visible in scenarios such as the biped with a rich sensory apparatus at a control frequency of $4\,\mathrm{Hz}$, as illustrated in Figure 3.4. However, it is also worth noting that there are instances where NE takes longer to converge on an effective solution when using RNN compared to other models. For example, this slower

	MLP	RNN	SNN	SNN-H	Tot.
MLP	–	0	0	0	0
RNN	7	–	9	5	21
SNN	3	1	–	0	4
SNN-H	3	0	4	–	7

Table 3.3: Number of cases (i.e., combinations of morphology, sensory apparatus, and control frequency) for which the search with the neural model on the row is statistically significantly more efficient (in terms of g_{80}, see text) than the search with the neural model on the column. The last column shows the overall number of pairwise comparison for which a neural model is statistically significantly better than one other neural model.

convergence is observed in the case of the biped with sight at a control frequency of $60\,\mathrm{Hz}$.

For what concerns the other models, Table 3.3 and Figure 3.5 show that (1) MLP never results in a faster search than the other models, and (2) homeostasis is beneficial also for efficiency: SNN-H is never worse than SNN and outperforms it in a few cases.

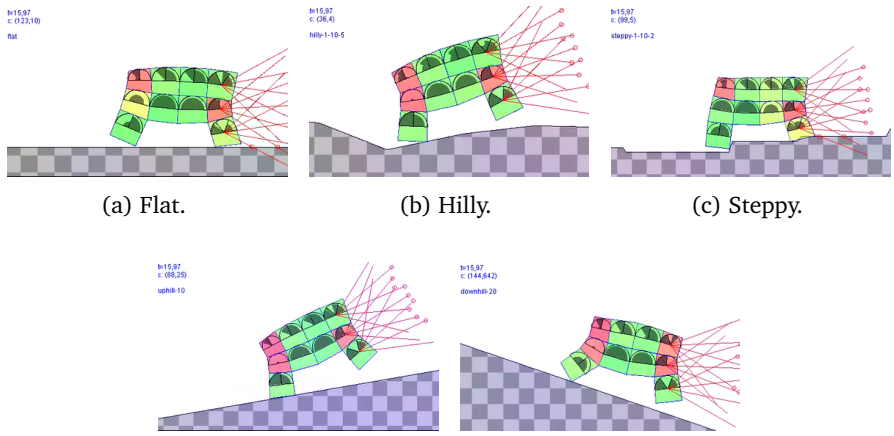
Beyond neural models, Figure 3.5 suggests that—not surprisingly—the number $|\boldsymbol{\theta}|$ of parameters does impact the search efficiency; in turn, $|\boldsymbol{\theta}|$ depends mainly on the number m of sensors in the VSR, as shown earlier in Table 3.1. For touch, that corresponds to $|\boldsymbol{\theta}| \in [24, 115]$, the search is in general very fast: 100 generations are often enough to find an effective solution. For the sight and rich sensory apparatuses, for which $|\boldsymbol{\theta}| \in [210, 625]$ and $|\boldsymbol{\theta}| \in [1575, 3521]$ respectively, the g_{80} is not as different as one could expect, given the rather large difference in $|\boldsymbol{\theta}|$. We hypothesize that this is motivated by the fact that searching for a good controller is harder with sight only: from another point of view, while the search space for sight is smaller than the one for rich, the fitness landscape for the former might be harder to be explored [416].

3.4.4 Generalization Ability

We define as generalization ability the ability of an evolved neural controller to retain its effectiveness when operating in conditions different from the ones it has been evolved in. For the task of locomotion, we consider the x-velocity as effectiveness and the shape of the terrain as conditions.

In particular, for measuring the generalization ability of the four neural models coupled with the different morphologies, sensory apparatuses, and control frequencies, we take the best VSR obtained at the end of each evolutionary run

(i.e., the one giving $v_x^\star$ for that run) and measure its x-velocity on a set of 16 *unseen* terrains, different from the flat one used for measuring the fitness. We consider uphill and downhill terrains with an even surface, and terrains with an uneven surface, including some with small steps and some with small hills (see some examples in Figure 3.6). We collect the 16 v_x values from the simulations (computed discarding the initial 5 s, as for the evolution) and average them, obtaining one $\bar{v}_x$ for each VSR. Finally, we compute the ratio $\rho = \frac{\bar{v}_x}{v_x^\star}$ between the average x-velocity on the unseen terrains and the one obtained on the flat terrain: ρ represents the degree to which the effectiveness in locomotion is retained in different conditions, hence, it represents the generalization ability of a neural controller—the greater the better, i.e., the more general the controller.



(a) Flat. (b) Hilly. (c) Steppy.

(d) Uphill. (e) Downhill.

Figure 3.6: Different types of terrains employed for measuring VSR adaptability.

Figure 3.7 shows ρ for the 72 different cases in the form of a matrix of boxplots, organized in the same way as Figures 3.3 and 3.5. By quickly comparing Figure 3.7 and Figure 3.3, it becomes clear that the variations among neural models are less pronounced for ρ than for $v_x^\star$. This observation is further validated by Table 3.4, which presents the results of a statistical significance analysis conducted similarly to the one for $v_x^\star$ in Tables 3.2 and 3.3.

On average, the x-velocity achieved by the best VSRs on unseen terrains is 40 % of what is achieved on flat terrain. Regarding generalization ability, RNN does not outperform the other neural models as clearly as it does for $v_x^\star$. Although VSRs with RNN are still the fastest on unseen terrains, ρ is relatively smaller. We believe this is due to the fact that the gaits optimized with RNN for flat terrain cause the corresponding VSRs to struggle more in different environments.

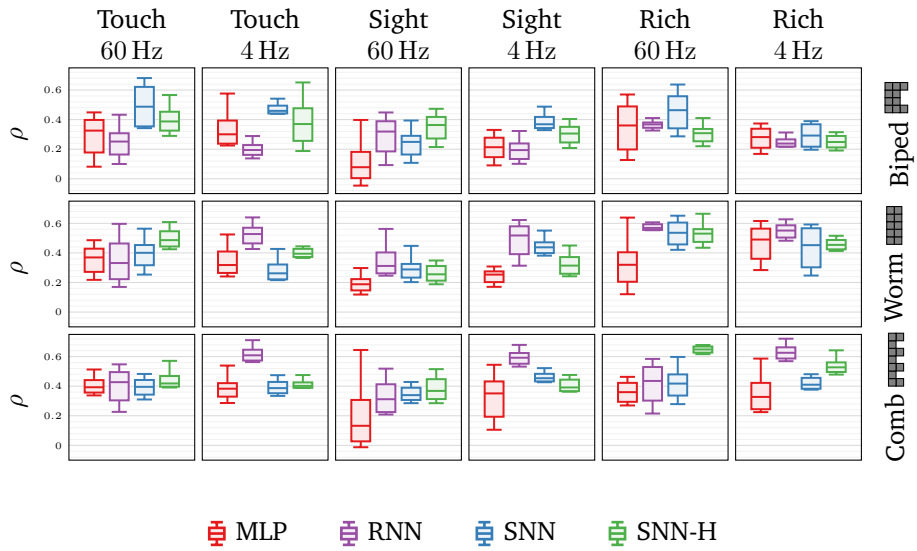


Figure 3.7: Generalization ability ρ of the evolved neural controllers.

On the other hand, MLP continues to be the least effective neural model. Table 3.4 indicates that it is outperformed by other models in the most cases. Additionally, despite achieving low $v_x^\star$ values when paired with sight at $f_c = 60\,\text{Hz}$, it remains the worst model in terms of ρ. Given that MLP is the only model without a state, this suggests that memory contributes to better generalization ability.

Finally, the differences between SNN and SNN-H in terms of ρ are minimal, as shown in Figure 3.7 and Table 3.4. This indicates that homeostasis offers a clear benefit in effectiveness without leading to a significant difference in generalization ability.

3.4.5 Behavioral Analysis

Besides the quantitative analysis devoted to assessing the performance of the neural models, it is relevant to characterize, on a more qualitative level, the behaviors each of them induces when coupled with different VSR morphologies and sensory apparatuses. Given the large amount of combinations considered, though, it is impossible to visually inspect all the evolved VSRs, yet, with an overview, we notice some core differences worth systematically analyzing.

To this extent, we rely on some behavioral features automatically extracted from the gait of the VSR in a simulation, based on the position and orientation of

	MLP	RNN	SNN	SNN-H	Tot.
MLP	–	0	0	0	0
RNN	7	–	4	4	15
SNN	2	3	–	2	7
SNN-H	4	4	2	–	10

Table 3.4: Number of cases (i.e., combinations of morphology, sensory apparatus, and control frequency) for which the neural model on the row is statistically significantly better (in terms of ρ, see text) than the neural model on the column. The last column shows the overall number of pairwise comparison for which a neural model is statistically significantly better than one other neural model.

its voxels over time [338]. In further detail, we proceed as follows:

1. for each i-th voxel of the VSR, we consider the signals $x_i^{(k)}, y_i^{(k)}, \beta_i^{(k)}$ of its center x- and y-coordinates and of its rotation β—for the latter, we consider the voxel initial (i.e., at $k = 0$) rotation as reference;
2. for each time step and each of the three signals, we compute the average across all the voxels, hence obtaining three VSR-wise signals $x^{(k)}, y^{(k)}, \beta^{(k)}$;
3. we compute the first differences of the signals, obtaining $\Delta x^{(k)}$, $\Delta y^{(k)}$, $\Delta \beta^{(k)}$;
4. for each of the three signals, we calculate the Fast Fourier Transform (FFT), from which we take the magnitude, filter out frequency components not in the range $[0\,\mathrm{Hz}, 4\,\mathrm{Hz}]$, and re-sample the remaining components to obtain 8 of them for each signal.

From the concatenation of these $3 \cdot 8$ components we obtain the final *behavior vector*, i.e., the feature vector describing the behavior of the VSR in a simulation.

We apply this feature extraction procedure to the gait of the best VSR obtained from each evolutionary run. As during evolution, we conduct a simulation of $30\,\mathrm{s}$ on a flat terrain, but we consider only the last $25\,\mathrm{s}$ for obtaining the behavior vector of the VSR, assuming the discarded $5\,\mathrm{s}$ are enough to reach a steady gait.

In order to enable the comparative visualization of the behavior vectors, we perform a dimensionality reduction using the PCA. In particular, we group the behavior vectors by VSR morphology and, after standardizing them, we rely on the PCA to perform a dimensionality reduction from 24 to 2. For the biped and the comb morphologies, the reduction captures around $60\,\%$ of the total variance, whereas for the worm, it captures around half of it. For each of the morphologies, we visualize the first two components obtained in a scatter plot, distinguishing

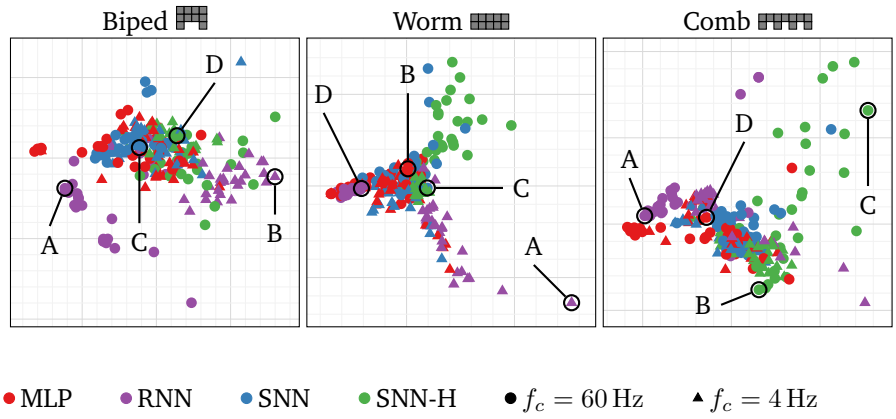
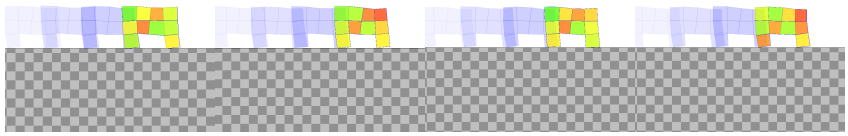


Figure 3.8: Behaviors PCA. Note that PCA was performed independently for each shape, i.e., the principal components are not the same in the plots. The letters in each plot refer to the individuals we sampled for visual inspection: the videos of the simulations are available online. For the biped, the four behaviors are shown in the form of time-lapses in Figure 3.9.

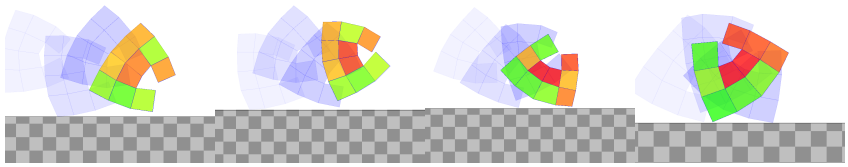
neural models with different colors, and control frequencies with different markers. We report the scatter plots for the three morphologies in Figure 3.8.

In each plot of the figure, the points are mostly clustered in one area with some scattered around. This suggests that each morphology is linked to a primary effective behavior or a range of similar behaviors, with some room for variability. For the biped morphology, the first plot shows significant overlap between MLPs and SNNs at the center. Moving away from the center, RNNs with $f_c = 60\,\text{Hz}$ are clearly separated on the left, while RNNs with lower control frequencies are more similar to SNN-H, slightly to the right. In the worm morphology plot, the main factor distinguishing the points appears to be the control frequency, with a large overlap in the center, higher frequency controllers in the top (with both types on SNNs), and lower frequency controllers in the bottom. Last, for the comb morphology, the central area is primarily occupied by MLPs and SNNs, with MLPs and RNNs with $f_c = 60\,\text{Hz}$ on the left and SNN-H scattered on the right.

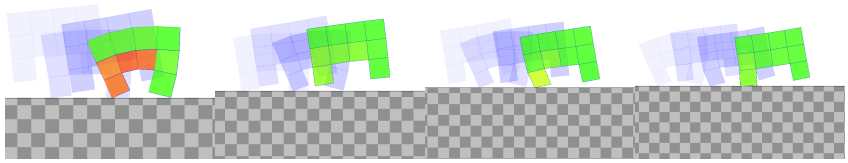
Looking at the global picture across all morphologies, RNNs and SNN-H generally move away from the center, indicating they tend to produce more diverse behaviors, especially in the worm and comb morphologies. Additionally, for these models, the control frequency has a more significant impact on VSR behavior compared to MLPs and SNNs. This suggests that a larger state makes control frequency more crucial in guiding behavior. This effect is harder to detect from



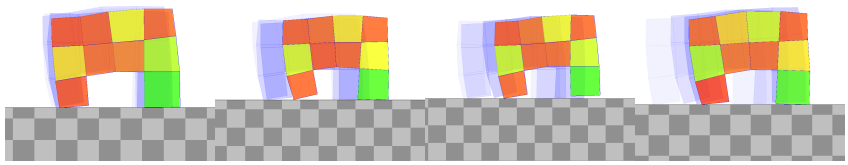
(a) Behavior A



(b) Behavior B



(c) Behavior C



(d) Behavior D

Figure 3.9: Time-lapses showing four behaviors of a biped VSR doing locomotion—namely, the shown behaviors correspond to named markers of the biped panel of Figure 3.8. Frames are taken at $1\,\mathrm{s}$ intervals (simulated time); in each frame, the blue shades show the position of the robots at earlier $0.5\,\mathrm{s}$ intervals. The color of each voxel represents the current area ratio with respect to the rest area (red for contracted, i.e., < 1, green for expanded, i.e., > 1, yellow otherwise, i.e., ≈ 1).

performance results alone, as evolution often compensates for less effective behaviors with finer parameter adjustments.

Finally, we manually select 4 evolved VSRs for each morphology that correspond to distinct points in the PCA plots of Figure 3.8, marked with capital letters in the figure. We examine the 12 behaviors, confirming that they are qualitatively different. The videos of these simulations are publicly available online[2], while we report the time-lapses showing the four behaviors of the biped in Figure 3.9. Among the selected VSRs, one for each morphology exhibits a vibrating behavior: A for the biped, D for the worm, and A for the comb. Notably, in all three cases, the VSRs are equipped with an RNN operating at $f_c = 60\,\text{Hz}$, as RNNs are prone to such behavior due to their structure, where hidden neurons receive their previous activation values. Although vibration is effective for locomotion in our experiments, real VSRs might not achieve effective locomotion through vibration alone. Therefore, RNNs seem to be more susceptible to the reality gap problem [372, 430] compared to SNNs and MLPs, especially at $f_c = 60\,\text{Hz}$.

3.5 Concluding Remarks

Choosing the most effective neural network models for robotic control remains an open question, especially for modular and soft robots due to their complex dynamics resulting from component interactions, morphological computation, sensory inputs, and environmental responses. In this chapter, we conducted a systematic experimental campaign with VSRs to explore the advantages of different neural network models, including those with and without state. Our main findings indicate that networks with recurrent connections are generally more effective and efficient. However, SNNs often exhibit better generalization to environmental changes. This is especially true when homeostatic plasticity is incorporated, allowing for rapid adaptation during the lifetime of the robotic agent.

[2] https://giorgia-nadizar.github.io/NeuralModelsVSR/

4

Collective Intelligence with Spiking Neural Cellular Automata

A VSR can be viewed as a collection of simple agents, the voxels, that work together to produce the overall VSR behavior. In this paradigm, collective intelligence is crucial for coordination, as each voxel operates independently, using only local sensory information and knowledge from its immediate neighbors. In this short chapter, we propose a new form of collective control influenced by Neural Cellular Automata (NCA) and based on bio-inspired SNN: the embodied Spiking Neural Cellular Automata (SNCA). We test various SNCA variants and find them to be competitive with state-of-the-art distributed controllers for locomotion tasks. Additionally, our results demonstrate significant improvements over the baseline in terms of adaptability to unforeseen environmental changes, which could be crucial for the physical viability of VSRs.

This chapter is based on the following publication: **G. Nadizar**, E. Medvet, S. Nichele, and S. Pontes-Filho. Collective control of modular soft robots via embodied Spiking Neural Cellular Automata. In *Workshop on From Cells to Societies: Collective Learning across Scales (Part of ICLR)*, 2022 [297].

4.1 Introduction and Related Works

Biological organisms exhibit intrinsic modularity at various scales [241]. Collective self-organization at the cellular level leads to the formation of complex bodies and brains without centralized control. This modularity enables local interactions, which facilitate collective learning and adaptation.

In the artificial domain, modular robotics [5] offers a framework for exploring biologically-inspired principles of collective control through the distributed coordination of the robot components [61]. Modular robots also enable high reconfiguration and self-assembly capabilities [329], along with fault tolerance and module reusability. To fully leverage these advantages, modular distributed controllers, possibly embedded in each module, are necessary. Thus, the robot overall behavior results from the collective interaction of distributed sensing, local communication, and actuation among its modules. Moreover, using identical modules enhances part reusability and robustness in case of damage [163]. Here we focus on VSRs, where mechanisms of collective intelligence are clearly beneficial. While such mechanisms are commonly explored in swarm robotics [142], such as self-assembly through local interactions [366], they are less investigated in the context of modular robotics.

One paradigm of distributed neural control through the local interactions of identical cells is NCA [225, 310, 286]. For robots composed of identical modules, each NCA cell could be implemented within a robot module. This approach could facilitate physical realization by eliminating the need for global wiring or centralized control. NCA have been successfully applied to grow and replicate Cellular Automaton (CA) shapes and structures using NE [310] and differentiable learning [286], to create self-organizing textures [312], to grow 3D artifacts [407], for self-classification [451], for regenerating soft robots [160], and even for controlling reinforcement learning agents [432].

In this short chapter, we build on the findings of Chapter 3 w.r.t. the benefits offered by biologically plausible SNNs to devise a novel embodied NCA model based on SNNs, which we call *embodied SNCA*. The SNCA can exploit the highlighted advantages offered by SNNs, such as self-regulatory mechanisms, i.e., homeostasis, and memory. Moreover, the structure offered by the embodied controllers and their communication channel provides intrinsic recurrence, which we observed to be beneficial in Chapter 3.

Additionally, nearby modules communicate through spikes, which occur only when the internal neural membrane potential reaches a specific threshold. This interaction alters the membrane potential of the post-synaptic neuron, either within the same module or in a neighboring module in the case of modular robots. The emergence of neuromorphic hardware, which supports SNNs execution and learning natively, may significantly enhance energy efficiency com-

pared to traditional ANNs [36] which is a key factor for the practical realization of self-organizing VSRs.

4.2 Spiking Neural Cellular Automata

The framework of SNCA is built upon two primary components: (1) distributed control with explicit communication for VSRs, and (2) SNNs.

The distributed controller with explicit communication, presented in Section 2.1.3, is closely related to NCA. Specifically, each voxel in the VSR functions as a cell in the NCA, with the controller acting as the ANN of the NCA.

We instantiate it in three variants: *non-uniform directional* (~~U~~D), *uniform directional* (UD), and *uniform non-directional* (U~~D~~), which differ in the homogeneity of the individual cells (uniform vs. non-uniform) and in the information passed between voxels (directional vs. non-directional). The most evident, yet conceptually simple, difference lies in the *uniformity*: in ~~U~~-NCA, cells have a different ANN in each voxel (each with parameters $\boldsymbol{\theta}_i$), whereas in U-NCA all cells ANNs share the same parameters $\boldsymbol{\theta}$. Therefore, it follows that, for a given ANN architecture, and for a VSR with n voxels, the amount of parameters of ~~U~~-NCA is n times the amount of parameters of U-NCA.

Among the considered variants the one which is more closely related to original paradigm of NCA is the U~~D~~-NCA, thus we consider this version for the proposed SNCA (i.e., U~~D~~-SNCA), whereas we leave all other combinations as baselines, considered in combination with MLPs.

Concerninf the SNN part of the SNCA, we employ the Leaky Integrate and Fire (LIF) model with homeostasis, outlined in Section 2.2.3. Consequently, each of the local controllers is implemented with an SNN, and communication between neighboring voxels is achieved through spikes transmission. However, when interfacing with sensors and actuators, we convert between spike trains and real-valued data as detailed in Section 2.2.4.

4.3 Experimental Evaluation

We perform an extensive experimental campaign to investigate how coordination can emerge from different forms of collective control. We aim at evaluating if we can improve the baselines of distributed control for VSRs [260] with our novel contribution, the U~~D~~-SNCA. Therefore, we address the following research question: *"are SNCA superior with reference to the baselines of distributed control?"*.

To determine the effectiveness of a collective controller, we deploy it onto a VSR, and optimize its parameters using as fitness measure the velocity achieved by the robot performing locomotion on a flat terrain. In addition, we also assess

the controllers adaptability, by measuring the VSR velocity, after the optimization, on a set of unseen terrains, i.e., terrains not used to optimize the controller parameters. With the extent of obtaining more general results, we experiment with three different morphologies.

4.3.1 Uniform Non-Directional SNCA vs. Baseline Embodied NCA

In order to provide an answer to the posed research question, we start by optimizing the parameters of three variants of NCA for each of the three considered morphologies, for a total of nine VSRs optimizations.

Concerning the NCA, we take into consideration the UD- and $\not{U}\not{D}$-NCA as baselines, and we compare them against the U$\not{D}$-SNCA. We use an MLP with $\tanh$ as activation function for both baseline NCA, while we equip the SNCA with a fully-connected feed-forward SNN based on the LIF neural model augmented with homeostasis, with the following parameters: $v_{\text{rest}} = 0\,\text{mV}$, $\lambda_v = 0.01/\text{s}$, $\vartheta^{(0)} = 1\,\text{mV}$, $\psi^{(0)} = 0\,\text{mV}$, $\psi_{\text{inc}} = 0.2\,\text{mV}$, $\lambda_\psi = 0.01/\text{s}$, $n_w = 5$ (we set $f_c = f_{\text{sim}} = 60\,\text{Hz}$). For both ANNs, we use 1 hidden layer, setting its size equal to the size of the input layer. We set $n_c = 1$ for both UD- and $\not{U}\not{D}$-NCA, and $n_c = 4$ for the U$\not{D}$-SNCA, in order to make the sizes of the ANNs output layers equal. Our choice of NCA hyper-parameters is driven by some exploratory experiments and by previous work involving SNNs [340, 299] and NCA [310, 286].

Regarding the morphologies, we experiment with 3 VSRs, a biped  , a comb  , and a worm  —the same ones considered in Chapter 3, with the exception of the worm which was thicker. We choose these morphologies to test the NCA controllers versatility, because they resemble different forms of living organisms, which take advantage of their diverse body shapes to achieve diversified gaits. We rely on 2D-VSR-Sim [262] for the VSRs simulation, leaving all parameters to their default values. The code for the experiments is publicly available online[1].

To optimize the NCA parameters, we resort to the ES of Algorithm 2.2, with $n_{\text{pop}} = 36$, $n_{\text{elite}} = n_{\text{pop}}/4 = 36$, $n_{\text{evals}} = 30\,000$, and $\sigma = 0.35$.

We optimize VSRs for the task of *locomotion* on a flat terrain, the goal being traveling as fast as possible along the positive x-axis. We assess the performance of a VSR by measuring its average velocity v_x along the x-axis during a simulation of $30\,\text{s}$. We discard the first $5\,\text{s}$ of each simulation to exclude the initial transitory from the velocity measurements. We use v_x as fitness measure for selecting the best individuals in the ES. For each of the 9 VSRs resulting from the combination of 3 NCA and 3 morphologies, we perform 10 independent evolutionary optimizations, i.e., with different random seeds, for a total of 90 runs.

[1] `https://github.com/giorgia-nadizar/VSRCollectiveControlViaSNCA`

Besides testing the VSR effectiveness upon parameters optimization, i.e., at the end of evolution, we also appraise their adaptability. We define a VSR controller as *adaptable*, if it is able to achieve good performance in locomotion in spite of environmental changes. To evaluate this in practice, we take each optimized VSR and re-assess it on a set of unseen terrains, i.e., terrains which none of its ancestors ever experienced locomotion on. In particular, we experiment with the following terrains (see some examples in Figure 3.6): (1) hilly with 6 combinations of heights and distances between hills, (2) steppy with 6 combinations of steps heights and widths, (3) downhill with 2 different inclinations, and (4) uphill with 2 different inclinations. As a result, we re-assess each individual on a total of 16 different terrains; we define its adaptability as the average of the v_x on those terrains (each computed in a $30\,\mathrm{s}$ simulation, discarding the initial $5\,\mathrm{s}$).

The results of our experimental evaluation are summarized in Figure 4.1. For each of the considered VSR morphologies and NCA variants, we display the distribution of velocities achieved at the end of evolution by the best individuals, and their performance in terms of adaptability, i.e., the distribution of their average velocity on unseen terrains. In addition, above pairs of box plots, we report the p-values resulting from a two-sided Mann-Whitney U statistical test; we consider, unless otherwise specified, $\alpha = 0.05$ as confidence level.

From Figure 4.1, we can see that for both the biped and comb morphologies, UD-SNCA consistently outperform the baseline in adaptability, and they show superior performance by the end of evolution in all but one instance. Conversely, the results for the worm morphology are notably different, presenting a challenging scenario for drawing a general conclusion to the research question.

To address this apparent contradiction, we analyze the behavior of several evolved VSRs[2] and identify a clear reason for the failure of the SNCA with the worm morphology. Specifically, we observe that NCA systems based on MLPs generate high-frequency dynamics, resulting in a vibrating behavior. This issue is mitigated in SNCA by homeostasis and by the choice of $n_w = 5$ for converting spikes to actuation values. However, for this worm morphology, vibration appears to be the only effective gait, as these VSRs lack the flexibility to bend properly, constrained by having only one row of voxels—note that the worm of Chapter 3, ▦, did not suffer from this problem being constituted by two rows of voxels. In contrast, the biped and comb morphologies feature more complex structures, enabling the discovery of a wider range of effective gaits. Indeed, our examination of these two morphologies reveals a broader variety of gaits, with some controlled by MLP-based NCA exhibiting tendencies toward vibration. Avoiding such vibrating behaviors is crucial, as they significantly impair adaptability and pose a major challenge to the physical practicality of VSRs, creating a form of

[2]Videos available at `https://giorgia-nadizar.github.io/VSRCollectiveControlViaSNCA/`.

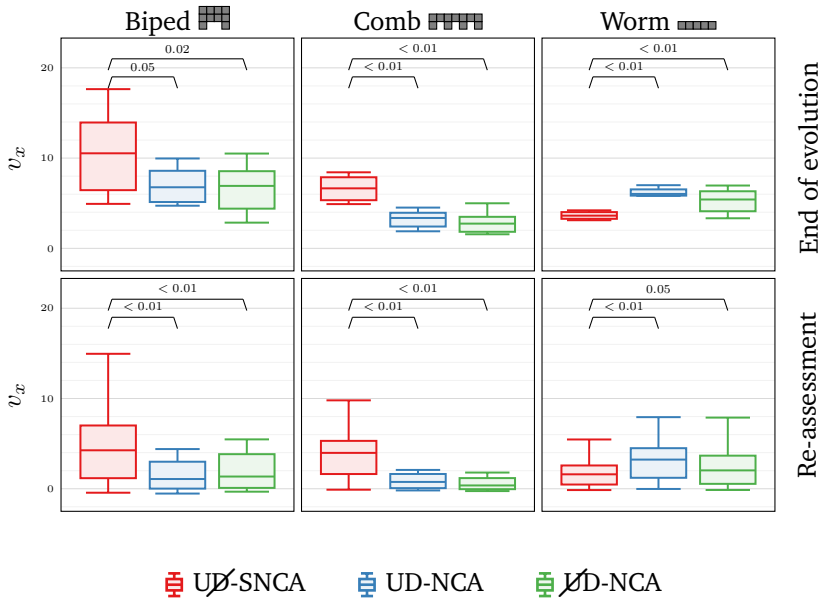


Figure 4.1: Box plots of the velocities v_x achieved by the best individuals at the end of evolution and upon re-assessment on unseen terrains for different VSR morphologies (plot columns) and embodied NCA controllers (color). Above pairs of boxes we report the p-values resulting from Mann-Whitney U tests with the null hypothesis of equality of the means.

reality gap [430, 372]. Although vibrating behaviors can be explicitly discouraged, such as by reducing actuation frequency, having a controller designed to inherently avoid them represents a notable achievement.

4.3.2 Strengths of the Uniform Non-Directional SNCA

From the experimental outcomes, however, another question arises: *"what are the reasons behind the success of the UD-SNCA?"*. Namely, does the improvement lay in the non-directionality of the NCA or in the SNN employed?

To address the newly emerged points, we deepen our analysis with a supplementary experimental campaign, encompassing new combinations of NCA architectures, neural models, and morphologies, for a total of 12 additional VSRs to be optimized. Regarding the morphologies, we experiment with the biped and the comb, discarding the worm for the reasons highlighted in Section 4.3.1. For what concerns the controllers, we extend the previous experiments by evaluating

all missing combinations of NCA architecture and neural models. Among the latter ones, we also include a SNN composed of LIF neurons for which we disable homeostasis, keeping the value of the threshold fixed throughout the simulation to its initial value $\vartheta_i^{(h)} = \vartheta_i^{(0)} = 1\,\mathrm{mV}$. For each of the 12 new VSRs we repeat the experimental pipeline of Section 4.3.1.

We display the results, together with the outcomes of the previous experiments, in Table 4.1. Each cell of the table reports the median of the velocities achieved by VSRs at the end of evolution and upon re-assessment, grouped by morphology; each row corresponds to a NCA architecture, whereas we put neural models on the columns. We color cells proportionally to the median of velocities in order to better convey the information.

	End of evolution						Re-assessment					
	Biped			Comb			Biped			Comb		
	S-H	S	M	S-H	S	M	S-H	S	M	S-H	S	M
UD	6.5	5.5	8.0	3.8	5.9	3.8	3.7	2.3	2.2	2.6	3.8	1.4
~~U~~D	10.2	7.0	7.9	5.8	2.8	3.4	5.5	2.9	3.4	3.5	1.4	1.2
U~~D~~	13.1	13.0	9.3	7.5	5.9	8.5	5.5	5.5	2.2	5.1	3.9	4.7

Table 4.1: Medians of velocities v_x achieved by the best individuals at the end of evolution and upon re-assessment on unseen terrains for different morphologies. We put different NCA architectures on each row, and ANN models on the columns (M stands for MLP, S for SNN without homeostasis, S-H for SNN with homeostasis). Cells are colored proportionally to v_x.

By examining Table 4.1, we can assess the significance of the two factors mentioned earlier. First, to evaluate the impact of the NCA architecture, we compare the medians of different rows across each column of the table. We find that U~~D~~-NCA perform no worse than D-NCA in all but one case, and they either match or exceed the performance of D-NCA when combined with SNNs. This may be due to the fact that, particularly in the absence of agent specialization (as seen with U-NCA), it is easier for a prototype individual to learn to send a single message to all its neighboring clones and correctly interpret the information received. Additionally, U~~D~~-NCA are inherently less likely to produce vibrating dynamics, making them more effective when paired with SNNs, which also benefit from this characteristic.

Regarding the importance of the neural model, we observe that SNNs, both with or without homeostasis, outperform MLPs in 10 out of 12 cases. To further evaluate the impact of homeostasis on SNNs, we focus on the re-assessment results, where this neural model consistently delivers superior outcomes in all but

one case, highlighting its critical role in self-regulation and adaptation. Additionally, it appears that SNNs are more naturally compatible with U$\not{D}$-NCA, as both are inclined to avoid high-frequency, non-adaptable behaviors. Therefore, we can conclude that the strength of our approach lies in the successful integration of the U$\not{D}$-NCA architecture with SNNs incorporating homeostasis.

4.4 Concluding Remarks

In this short chapter, we investigated the paradigm of collective control in VSRs, focusing on how coordination emerges from the combined actuation of individual agents, or voxels, within the VSR. Drawing inspiration from NCA, a distributed neural control method, and embodied control techniques for VSRs, we proposed the novel concept of embodied SNCA, utilizing SNNs as its fundamental units. To assess the performance of the proposed SNCA as a robotic controller, we compared it with state-of-the-art embodied controllers, optimizing the parameters for three different VSRs in locomotion tasks. Our experimental results showed that the SNCA not only matches the performance of existing controllers but also significantly enhances adaptability, outperforming competitors in unforeseen situations. Furthermore, we observed a trend towards behaviors with reduced reality gaps in VSRs controlled by SNCA, suggesting a promising path toward the physical feasibility of these robots.

5

Specialization through Plastic Totipotent Neural Controllers

Multi-cellular organisms typically originate from a single cell, the zygote, that then develops into a multitude of structurally and functionally specialized cells. The potential of generating all the specialized cells that make up an organism is referred to as cellular "totipotency", a concept introduced by the German plant physiologist Haberlandt in the early 1900s. To emulate this process in synthetic organisms, we present a model based on VSRs, where both the robot structure (the arrangement of voxels) and its control system (an ANN) are optimized for the task of locomotion. We introduce Hebbian learning to allow the ANNs to adapt their weights during movement, creating an analogy between totipotent cells and totipotent ANN-controlled modules. Our in silico experiments show two key results. First, Hebbian plasticity improves performance and adaptability. Second, and more importantly, we find that plasticity enables modules to specialize functionally and behaviorally, despite having the same initial ANN. This specialization mimics totipotency at the ANN level and has potential implications for AI and applications thereof, such as modular robotics and multi-agent systems.

This chapter is based on the following publication: A. Ferigo*, G. Iacca*, E. Medvet*, and **G. Nadizar*** (* equal contribution). Totipotent Neural Controllers for Modular Soft Robots: Achieving Specialization in Body-Brain Co-Evolution through Hebbian Learning. *Neurocomputing*. 2024 [113].

5.1 Introduction

One of the most remarkable features of multicellular organisms is that they typically originate from a single cell, which, starting with a common "code" known as the genome, develops into a variety of structurally and functionally specialized cells. The ability of non-reproductive cells, known as stem cells, to generate all the specialized cells that constitute an organism through repeated divisions is generally referred to as cellular "totipotency". This concept was introduced by the German plant physiologist Haberlandt in the early 1900s and has since been expanded upon in several subsequent works.

Thanks to this mechanism, stem cells can develop into any type of cell in the body, from muscle fibers to various blood cells, and ultimately into neurons within the nervous system. It is important to note that this cellular specialization is not the only factor contributing to the success of multicellular life forms. Structurally specialized cells can also become behaviorally specialized, meaning they can adapt their behavior over their lifetime. For example, muscle cells adjust their behavior based on the type of training they undergo [164] (e.g., endurance versus strength training), white blood cells remember threats to our system and adapt their response accordingly [353], and neurons reorganize their connections through learning, creating new synapses via the process of synaptogenesis [333].

In light of these observations, it is evident that both structural and behavioral adaptation—across various spatial scales, from cells to the entire body—is crucial to an organism's success. Moreover, adaptation occurs on multiple temporal scales, playing a vital role both during an organism's lifetime and beyond it, through the process of evolution. Eiben and Hart [99] have recently encapsulated the idea of adaptation during and beyond an individual's lifespan with a single concept: "if it evolves it needs to learn". While this concept was introduced specifically in the context of morphologically evolving robotic systems [247], it can arguably be applied to other domains as well, including biology, where evolution and adaptation/learning interact.

In this work, we build upon this concept by specifically exploring the interaction between *specialization*, *learning*, and *evolution*, with a particular focus on body-brain co-optimization. We aim to replicate the totipotency mechanism in VSRs [153], where both the body (the arrangement of voxels) and the brain (the ANN controlling each module) undergo a co-evolutionary process to optimize the robot locomotion capabilities. The modularity of these robots offers an excellent opportunity to replicate and study the specialization mechanism *in silico*. We hypothesize that for *modular* evolving entities, the concept of "if it evolves, it needs to learn" should be expanded to "if it evolves and learns, it needs to specialize *at the level of its modules*". Essentially, we draw a parallel between totipotent

cells in living organisms and totipotent ANN-controlled modules, where specialization is achieved through Hebbian learning [49]. This learning process allows the ANNs weights to adapt differently for each module during the execution of the locomotion task.

Thus, this paradigm unifies into a single model the following key elements: (1) modularity (and hence specialization at the level of modules), (2) learning (particularly, Hebbian learning), and (3) evolution (particularly, body-brain co-evolution). Previous studies [99, 145, 337] have demonstrated that learning enhances the body-brain co-evolution of robots, likely because adaptation during an organism's lifetime enables the exploration of a broader search space compared to non-plastic agents—an example of the Baldwin effect [19, 215]. In this chapter, we extend this idea by allowing learning to operate *independently* within the robot individual modules, where different modules must cooperate to achieve a common goal. Importantly, since the modules in our robots are identical in terms of mechanical properties, structure, and the initial weights of their ANN-based controllers, Hebbian learning enables us to isolate the effects of specialization (i.e., determining whether and how identical modules behave differently) from those of the co-evolution of the robot body and brain.

Furthermore, it is important to highlight that Hebbian learning is a form of *unsupervised learning*: during their lifetime, the ANNs controlling the robot modules are not guided by an externally imposed reward signal. Instead, they adapt according to emergent rules that result from the evolutionary process. In this context, our ad hoc experimental analyses reveal that the differentiation of the modules—specifically regarding their position and access to sensory information within the robot body—is crucial for driving adaptation through Hebbian learning.

5.2 Related Works

In the following, we briefly review the literature related to the three main concepts mentioned earlier which are used in our work, namely (1) specialization, (2) learning, and (3) body-brain co-evolution.

5.2.1 Specialization

As previously mentioned, a totipotent cell is a type of stem cell capable of differentiating into any other cell type. It is important to note that neurons can undergo further behavioral specialization, a process known as *neural differentiation* [162], allowing them to become functionally specialized based on their location in the brain and, more broadly, within the nervous system.

In the realm of ANNs, various studies have explored a similar type of specialization at the *neural level*, whether in the context of a single task, such as object recognition [91], or in a single ANN designed for multiple tasks [465]. These studies arrived at the same conclusion: following training, the ANN becomes organized into functionally specialized regions.

Outside the domain of ANNs, a recent study demonstrated that in Multi-Agent Reinforcement Learning (MARL) scenarios where the goals of a group of agents are *aligned*, specialization facilitates cooperation [253]. Similar insights were discussed in [464], where the authors found that in public games, social sanctioning can be used to manage specialization within a group of agents. Additionally, Kosak et al. [198] proposed a mobile multi-robot system in which agents can autonomously reconfigure themselves, effectively achieving a form of totipotency. This study primarily focuses on the hardware and system design enabling robotic reconfigurability, rather than the adaptation mechanisms that lead to specialization, which is the focus of this chapter.

In the field of modular robotics, Auerbach and Bongard [15] explored how different body parts of an agent could specialize to handle a task consisting of two sub-tasks, demonstrating how the modules could become specialized for one or both sub-tasks. More recently, Whitman et al. [457] investigated specialization in modular soft robots. However, that study differs from ours in that the modules were manually designed and then assembled to create specific specialization patterns within the overall agent.

In summary, while specialization has been extensively studied in Multi-Agent Systems (MASs) and at the module level in modular robots, and although some research has investigated specialization at the neural level (i.e., specialization of specific neurons or groups of neurons within a given ANN), to our knowledge, no previous studies have examined the specialization of a *single* ANN into multiple ANNs for controlling modular robots. This aspect is a unique feature of our model. It is important to note that this type of specialization differs from the scenario where a single ANN learns to perform multiple tasks, which is often associated with the issue of "catastrophic forgetting" [192, 477], where the ANN tends to lose the ability to perform previously learned tasks when specializing in new ones.

5.2.2 Learning

It is a well-established principle in evolutionary theory that learning *during* an agent lifetime—meaning adapting the agent brain (or controller) while executing a given task—is crucial for achieving better results compared to relying solely on evolutionary approaches, particularly in complex scenarios. The significance of learning has been recognized in both natural evolution [19] and artificial evo-

lution [155, 224]. In both contexts, learning involves altering how the agent's brain processes information from the environment throughout its lifetime (or, in the case of artificial evolution, during task execution). This ability is known as *neural plasticity*, which can be categorized into structural or functional plasticity [385].

In the former case, plasticity influences the structure of the ANN, such as by pruning certain connections [156] or by developing the ANN from the ground up [306].

In the case of functional plasticity, instead, only the parameters of the ANN are altered while its structure remains fixed. This parameter adjustment is typically managed through plasticity rules, with Hebbian learning being the primary method employed in this work (see Section 2.2.1). Plasticity rules can also be encoded indirectly, as seen in Adaptive HyperNEAT [361], where a smaller ANN learns both the weights and the plasticity rules for the main controller. Other methods involve directly optimizing the parameters of fixed plasticity rules [272, 305, 463] or evolving entirely new functions [179]. Recently, two approaches have focused on reducing the number of plasticity rules or parameters in an ANN, either by clustering rules [331] or by grouping rules by neuron [112].

Finally, some approaches integrate both structural and functional plasticity. For instance, SBNN [106] draws inspiration from synaptogenesis by employing both a pruning algorithm and a plasticity mechanism to gradually adapt the ANN throughout the agent life. Additionally, Dresp-Langley [98] offers a survey that explores various biologically inspired learning models for robot control; this work is a valuable resource for further details and insights on the topic.

5.2.3 Body-Brain Co-Evolution

Body-brain co-evolution, where the body is typically defined by the mechanical parameters of the robot and the brain is represented as an ANN, has been extensively investigated in ER. The underlying premise is that co-evolving (or co-optimizing) both the brain and body of a robot reduces the bias and effort associated with human-designed systems. This approach is especially significant in modular robotics, where the morphological search space can be vast [471] and challenging to explore. Recent research on the fitness landscape of morphologically evolving robots [416] has highlighted the importance of selecting appropriate representations to achieve high-quality outcomes when optimizing both body and brain simultaneously.

Another benefit of co-evolving a robot brain and body is the ability to balance morphological and controller complexity [145, 175, 449], as opposed to evolving the morphology or the controller separately [317, 141]. Notably, Pagliuca and Nolfi [327] have emphasized that the enhanced performance of co-evolution

often stems from the ability to co-adapt morphological traits to control traits and vice versa. However, Mertan and Cheney [268] have pointed out that body-brain co-evolution might sometimes lead to premature convergence on sub-optimal solutions.

Indeed, using a single, centralized controller, as commonly done in body-brain co-evolution studies, can result in a mismatch between the ANN structure and the robot body [99]. A potential solution is to employ a *modular* controller, where distinct controllers are assigned to operate each module of the body. This approach simplifies the adaptation process by reducing the complexity associated with fitting a single controller to a new body shape and the number of parameters that need to be optimized [163, 210]. This is the strategy we utilize in our work.

5.3 Body-Brain Co-Evolution of VSRs

We optimize the body and the brain of a VSR at the same time, to make the robot effective at solving a given task, in this case locomotion. For the optimization we rely on the ES detailed in Algorithm 2.2, leveraging the representation presented in Section 5.3.1. Namely, we represent each candidate solution as a numerical vector $\boldsymbol{g} \in \mathbb{R}^p$, which describes both the body and the brain of the corresponding VSR. A similar approach has already been applied to concurrently optimize the body and the brain of a VSR [263, 110].

We measure the performance of the VSRs in a *locomotion task*, a traditional task in evolutionary robotics [316, 39, 94]. In the locomotion task, the VSR has to move as fast as possible along the positive x-direction on a flat terrain. To measure the degree of accomplishment of the task, we use the average velocity of the VSRs in a simulation lasting t_{final} time steps, computed considering the center of mass of the VSR and discarding the initial t_{init} time steps in which the robot might exhibit a transitory behavior.

5.3.1 VSR Representation

As already briefly mentioned, we represent a VSR as a numerical vector $\boldsymbol{g} \in \mathbb{R}^p$, which describes both its body and its brain. In particular, $\boldsymbol{g} = \left[\boldsymbol{g}_{\text{body}}\ \boldsymbol{g}_{\text{brain}}\right]$ is the concatenation of a vector $\boldsymbol{g}_{\text{body}}$ describing the body, and a vector $\boldsymbol{g}_{\text{brain}}$ describing the brain.

For the brain, we employ the distributed controller with explicit communication presented in Section 2.1.3, relying on either an MLP or an Hebbian MLP (H-MLP) as processing component (see Section 2.2.1).

We remark that it would also be possible to realize a similar study with SNNs, with Spike-Timing-Dependent Plasticity (STDP) as Hebbian learning [57], or with RNNs [50]. Yet, having observed the tendency of these ANNs to compensate

the effects of other aspects (e.g., sensory apparatus or morphology) in Chapters 3 and 4, we choose to limit this study to MLPs and H-MLPs.

For both cases, we consider a distributed architecture where all local controllers share the same parameters. Thus, for the brain part of the encoding, we simply set $\boldsymbol{g}_{\text{brain}} = \boldsymbol{\theta}$, i.e., we directly encode the parameters of the ANNs constituting the brain. We recall that $\boldsymbol{\theta}$ contains the synaptic weights for the MLP and the values of the ABCD coefficients for the H-MLP.

For the body, we use an indirect generative representation, as done by Ferigo et al. [110] and inspired by Cheney et al. [60]. Let $h \times w$ be the size of the largest body that can be represented: since a VSR body is a polyomino, it can be seen as a Boolean matrix $\boldsymbol{M} \in \{\text{true}, \text{false}\}^{h \times w}$ of $h \times w$ elements in which each element $m_{x,y}$ encodes the presence or absence of a voxel at position x, y. We use an ANN to fill $\boldsymbol{M}$ and directly encode its parameters $\boldsymbol{\theta}$ in $\boldsymbol{g}_{\text{body}}$. Namely, we use an MLP with 2 inputs and 1 output and compute each $m_{x,y}$ as:

$$m_{x,y} = \begin{cases} \text{true}, & \text{if MLP}_{\boldsymbol{\theta}}\left(\left\lfloor \frac{x}{w} \right\rfloor, \left\lfloor \frac{y}{h} \right\rfloor\right) \geq \tau \\ \text{false}, & \text{otherwise} \end{cases}, \tag{5.1}$$

where τ is the median value given as output by the MLP when applied to all the elements of the matrix. Finally, to ensure that the body is a proper polyomino, we only consider the largest portion of adjacent 'true' values in $\boldsymbol{M}$.

Concerning the sensors, we equip all modules with area, touch, and vertical and horizontal velocity sensors.

5.4 Experimental Evaluation

We perform an experimental evaluation aimed at answering the following two questions:

1. Does Hebbian learning have a positive impact on the VSR body-brain co-evolution? Are the resulting VSRs more effective than those without Hebbian learning?
2. Why is Hebbian learning beneficial? Does its effectiveness depend on how H-MLP-based controllers leverage specialization?

To address these questions we conduct a two-fold analysis. First, we perform the body-brain co-evolution (as detailed in Section 5.3) of several VSRs, optimizing them for the task of locomotion, and comparing the performance obtained by MLPs and H-MLPs. Then, we examine the evolved VSRs and their controllers, looking for motivations for the difference in performance between those equipped with MLPs and those using H-MLPs; in particular, we look for signs of specialization of the controllers embedded in different voxels.

5.4.1 Effectiveness of Hebbian Controllers

We perform 30 independent runs of the ES described in Algorithm 2.2 with $n_{\text{pop}} = 40$, $n_{\text{elite}} = n_{\text{pop}}/4 = 10$, $n_{\text{evals}} = 25\,000$, and $\sigma = 0.35$. We set the values of these parameters, and those of the following parameters specific to the task, based on the results obtained in previous experiments with this kind of robots [110, 109]. Concerning the task, we set $t_{\text{final}} = 180\,\text{s}$ and $t_{\text{init}} = 5\,\text{s}$. Concerning the representation, we set $w = h = 10$ (i.e., the evolved VSRs are at most 10×10 voxel large) and we use, for both the MLP and H-MLP controllers, a fixed architecture with one single hidden layer with the same size as the input layer. The latter choice results in having $(4 + 4 + 1) \times (4 + 4) + (4 + 4 + 1) \times 5 = 117$ synapses (given $|\boldsymbol{r}|+4$ inputs, $+1$ for the bias, i.e., $w_{l,i,0}$, and 5 outputs) in both types of ANN, hence giving $p = |\theta| = 117$ parameters to optimize for the MLP and $4 \times 117 = 468$ parameters for the H-MLP. We use the same architecture also for the MLP employed in the indirect representation for the body, resulting in other $(2 + 1) \times 2 + (2 + 1) \times 1 = 9$ parameters. In all cases we use $\tanh$ as activation function inside the ANNs. Therefore, overall we perform the body-brain optimization in $\mathbb{R}^{126}$ for the MLP case and in $\mathbb{R}^{477}$ for the H-MLP case.

Since the parameter η (the learning rate, see Equation (2.20)) plays a key role in determining the adaptation of the H-MLP-based controllers, we perform the experiments with two values: 0.01 and 0.1. We choose these two values as they are typically employed in similar studies, as, e.g., [331] or [109], and we deem them representative of two different learning dynamics (i.e., a slower and a faster learning process).

We execute all the experiments using the 2D-VSR-Sim simulator [261], setting $f_c = 4\,\text{Hz}$ and leaving all other parameters to the default values. The choice of such a control frequency derives from the considerations of Chapters 3 and 4 concerning the vibrating behaviors observed in VSRs with high control frequencies: we choose a lower value to explicitly prevent them.

Figure 5.1 summarizes the results of these experiments. It shows the fitness v_x of the best individual in the population during the evolution, one line for each of the three variants: MLP (that we use as a comparison baseline), H-MLP with $\eta = 0.01$, and H-MLP with $\eta = 0.1$.

The first observation from Figure 5.1 is that the adaptation facilitated by Hebbian learning positively impacts the body-brain co-evolution of VSRs. Robots equipped with H-MLPs are, on average, faster than those with MLPs, regardless of the value of η. Notably, this result is achieved using the same EA and the same number of fitness evaluations, despite the significantly larger search space in the case of H-MLP ($\mathbb{R}^{477}$ vs. $\mathbb{R}^{126}$). For each pair of variants, we perform a statistical significance test—a one-sided Mann-Whitney U rank test—after verifying the necessary assumptions. The null hypothesis is that the distribution of v_x for the first variant is statistically lower than or equal to that of the second variant. We

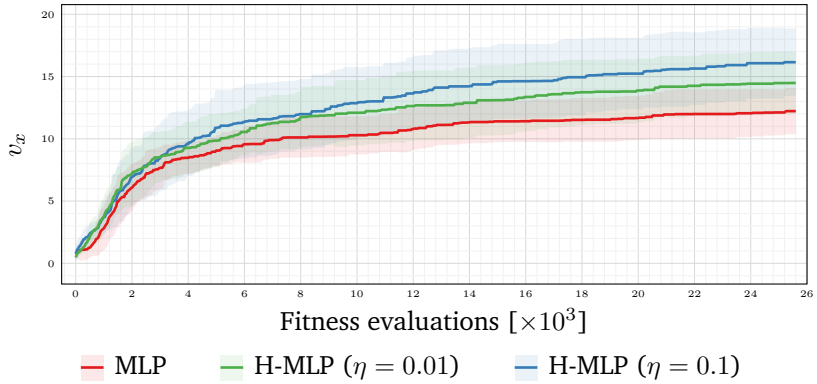


Figure 5.1: Median $\pm$ standard deviation (across 30 runs) of the velocity v_x of the best individual during the evolution. VSRs equipped with H-MLPs are in general faster than those equipped with MLPs.

obtain a p-value of 0.0007 for MLP vs. H-MLP ($\eta = 0.01$), 5×10^{-7} for MLP vs. H-MLP ($\eta = 0.1$), and 0.01 for H-MLP ($\eta = 0.01$) vs. H-MLP ($\eta = 0.1$), meaning that all differences are statistically significant.

We attempt to explain the superior effectiveness of robots using Hebbian learning by examining the body-brain coupling. As Medvet et al. [263] previously noted, it is challenging to find a distinct controller for each module of a modular robot while simultaneously optimizing the body [234]. This challenge is even more pronounced for VSRs, where the softness of the body significantly influences behavior, making the brain more sensitive to changes in the body [258]. We speculate that the H-MLP ability to adapt synaptic weights based on the information processed by the ANN enables initially identical H-MLPs to differentiate across various voxels. This differentiation likely helps the robots better manage body variability during evolution. We explore this hypothesis further in Section 5.4.2.

To gain more insights into the results summarized in Figure 5.1, we analyze more in detail the behaviors of the 30 best VSRs obtained for each of the three variants. In particular, we compute their average velocity in different subsequent phases of their "life", which, we recall, lasts $180\,\mathrm{s}$: namely, four periods lasting $30\,\mathrm{s}$. The result of this analysis is summarized in Figure 5.2 which shows, in the form of box plots, the distribution of the average velocity v_x for the three variants in the four phases.

There are two key observations we can make from Figure 5.2. First, VSRs equipped with H-MLPs demonstrate learning, meaning they adapt and improve over time during their lifespan. Specifically, their average velocity v_x during

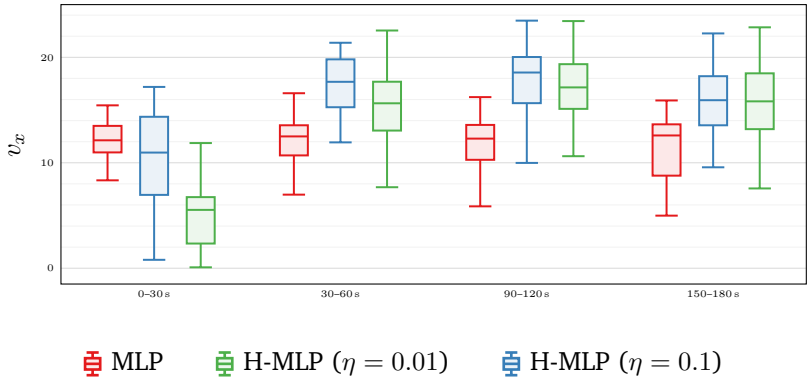


Figure 5.2: Distributions (across 30 runs) of the average velocity of the best robots during four consecutive intervals in the task (each lasting 30 s). VSRs equipped with H-MLPs do learn: they are faster after 30 s than at the beginning. With a lower learning rate η, they take longer (w.r.t. greater η) to become equally fast.

the first 30 s is noticeably lower than in the later phases. Additionally, a slight improvement can be observed between the 30–60 s and 60–90 s phases. This type of adaptation is not visible in VSRs equipped with MLPs.

Second, while the variant with $\eta = 0.01$ exhibits slower learning, as expected, the final velocity achieved by H-MLP-equipped robots is roughly the same, regardless of the η value. Consequently, the difference between the two lines in Figure 5.1 corresponding to the H-MLP variants is more attributable to the speed of learning rather than the overall effectiveness of the search.

Regarding the bodies, Figure 5.3 presents a subset of the evolved solutions, specifically those corresponding to seeds 1 to 10. Visually, there appear to be no significant differences between them. For instance, the "stair" shape emerges in all three settings: in the 5-th instance for MLP, the 9-th instance for H-MLP ($\eta = 0.1$), and the 1-st instance for H-MLP ($\eta = 0.01$).

We further analyze the bodies using the same descriptors introduced in [263], focusing on *elongation* and *compactness*. Elongation intuitively measures the ratio between the width and height of the body, while compactness describes the extent to which the body contains empty spaces. We perform a two-sided Mann-Whitney U rank test, confirming that there are no statistical differences for the descriptors used[1].

In the next section, we delve into an additional set of experiments and analyses conducted to uncover the reasons behind the superior performance of H-

[1]Videos of VSRs performing locomotion are available at `https://tinyurl.com/4ye6nkpu`.

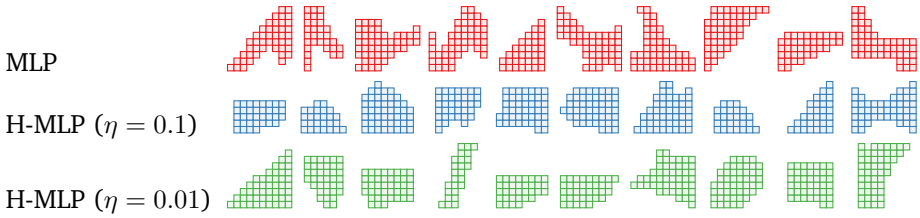


Figure 5.3: Samples of bodies evolved with the three variants (for seeds 1 to 10). There are no apparent differences between variants in terms of the shape of evolved bodies.

MLP-equipped robots compared to those that do not utilize Hebbian learning.

5.4.2 Specialization in Hebbian Controllers

Having observed a neat performance increase with the introduction of Hebbian learning, we conduct additional analyses to investigate the deep causes of such an improvement. In fact, we speculate that, since Hebbian learning confers each voxel the capability to learn based on its experience, this could lead to specialization, which in turn could facilitate the evolution and enhance the overall VSR performance.

To test our hypothesis, we perform two experiments on the evolved VSRs: an online one and an offline one. In the former one, we observe the H-MLP embedded in the voxels and analyze them while working inside the robots; in the latter, we pull the H-MLP out of the voxels and then perform further analyses. Namely, we detach the H-MLPs from the simulator and study their behavior using artificially generated inputs. For simplicity, we perform these experiments on a subset of 10 (out of 30) VSRs evolved in the experiment described in the previous section, due to the significant computational cost needed for the analysis. Namely, we select those corresponding to seeds 1 to 10 (i.e., those for which the body is shown in Figure 5.3).

Online Analysis

Our first analysis aims at verifying that: (1) learning is actually occurring in Hebbian controllers, and (2) there is some form of specialization in the voxels guided by the learning process. By "learning is actually occurring" we mean that Hebbian learning is guiding the weights towards values that are able to lead the VSR to the accomplishment of its task, even if no additional learning is performed; this is in contrast with an alternative hypothesis according to which the VSRs runs successfully only because of the dynamics induced by Hebbian learning. By

specialization, we mean that voxels ANNs become *functionally different* from one another after the learning process.

To test these two hypotheses, for each of $10 + 10$ VSRs equipped with H-MLP (10 for each value of η), we proceed as follows. First, we perform a simulation of $60\,\mathrm{s}$ where we let the VSR learn according to its evolved Hebbian rules. Then, we consider three different re-assessment conditions, namely:

1. *With learning*: in this case, for each seed, we clone the resulting VSR (with its learned weights) and we re-assess it in another $60\,\mathrm{s}$ simulation, where we allow the VSR to continue learning.

2. *Without learning*: as in the previous setting, but in this case we disable Hebbian learning in the new $60\,\mathrm{s}$ simulation, i.e., we "freeze" all the weights to their values reached after the initial learning phase.

3. *Homogeneous controller*: in this case, for each seed we copy the weights of one of the voxels ANN into all the other ANNs, obtaining a homogeneous controller. We repeat this evaluation for each of the voxels ANNs, hence obtaining as many clones as the number of voxels. Each of these clones is then re-assessed in another $60\,\mathrm{s}$ simulation, disabling Hebbian learning.

The results of this online analysis are presented in Figure 5.4, where we show the velocities v_x achieved by each evolved VSR in the three aforementioned conditions. In the first two settings ("With learning" and "Without learning"), each original VSR (one per seed) is reassessed, resulting in a single velocity value per subplot. However, in the third setting ("Homogeneous controller"), we obtain a distribution of velocities for each seed. The number of values in this distribution corresponds to the number of voxels in the VSR.

By comparing the first two velocities in each plot (i.e., those achieved with Hebbian learning enabled and disabled), we can draw conclusions about the occurrence of actual learning. In nearly all cases, except for one ($\eta = 0.01$, 8-th robot), the two values are close to each other. This suggests that after the initial learning phase, the weights of each ANN have converged to values that effectively control the VSR, making further Hebbian learning unnecessary, though it does not negatively impact performance. Therefore, we can confirm the occurrence of *actual* learning.

Turning to our second hypothesis, the one concerning specialization, we can draw conclusions by examining the rightmost distribution in each subplot. These distributions reveal two main scenarios: one with significant spread and another with less spread. In the first scenario, where distributions show considerable spread, the velocities range from very low values, indicative of nearly idle VSRs, to values comparable to those achieved without any controller alterations. This suggests that some voxels ANNs have specialized to such an extent that they are

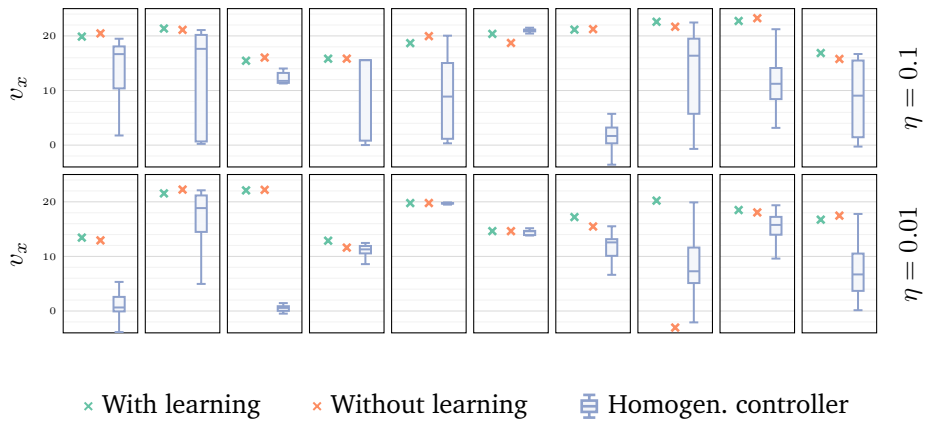


Figure 5.4: Velocities measured during the re-assessment of the evolved VSRs: one row per learning rate η, one column per VSR (seeds 1 to 10), one color per re-assessment condition.

not effective for controlling other voxels, leading to a significant performance decline when used in a homogeneous controller. On the other hand, the upper end of these distributions shows that for some voxels, specialization is less pronounced, allowing the local ANNs to remain versatile enough to control all voxels in the VSR effectively.

Regarding the second case, with denser distributions, we can still differentiate between high and low specialization. If the distribution is centered around high-velocity values, it indicates low specialization, meaning that all ANNs are versatile. Conversely, if the distribution is centered at low velocities, it suggests high specialization, where most ANNs are effective only for the specific voxel they are learned for. It is important to note that we do not enable learning when re-assessing the VSRs in the homogeneous controller condition. The rationale behind this choice is to assess the impact of having an ANN operate under different conditions compared to its specialized environment. Enabling learning might allow the ANN to re-adapt to the new conditions, which could complicate the interpretation of the results.

In summary, we can confirm that some voxels in the H-MLP have indeed specialized. However, this specialization is not uniform—neither across different VSRs nor within the voxels of a single VSR, despite all voxels being governed by the same Hebbian rules. The extent and uniformity of specialization might depend not only on the specific Hebbian rules configuration achieved by the end of evolution, but also on the distinct experiences encountered by each voxel (and thus each H-MLP embedded in it). To explore this further, we conduct an offline

analysis, which is detailed in the following section.

Offline Analysis

Building upon the findings of the previous section, we speculate that specialization in a voxel could be driven by its "experience", i.e., by the stimuli to which a given H-MLP is subjected while it is learning, in conjunction with the set of evolved learning rules. To test this hypothesis we conduct an offline analysis, aimed at: (1) assessing how different the stimuli of different voxels are, and at (2) evaluating how diverse the H-MLPs become in terms of responses to the same set of stimuli.

As a preparatory phase for our analysis, we gather the $10 + 10 + 10$ VSRs obtained with the experiment of Section 5.4.1 (again, in particular, those related to seeds 1 to 10), in this case both with MLP and H-MLP controllers, and simulate them for $60\,\mathrm{s}$. During these simulations, at each control time step, we collect and save the input vector of each ANN of the VSR. Moreover, for the VSRs controlled by H-MLPs, we also save a clone of the VSR with its achieved weight configuration at $t_s \in \{0.25, 15, 30, 45\}$. Following this data collection phase, we proceed with the offline analysis.

To tackle the first point (difference in stimuli), we start by examining the collected inputs, i.e., the various stimuli to which the different ANNs are subjected. We recall that at each time step h the input vector of an ANN, $\boldsymbol{x}^{(h)} \in [0,1]^8$, is defined as the concatenation of the sensor readings ($\boldsymbol{r}^{(h)} \in [0,1]^4$) and the communication values coming from its four direct neighbors.

To evaluate how different the experiences of the voxels of the same VSR are throughout their lifetime, we rely on the inputs distance matrix $\boldsymbol{D}$. For each VSR, we compute $\boldsymbol{D}$ as:

$$\boldsymbol{D} = d_{i,j} = \frac{1}{n_{\text{ts}}} \sum_{h=0}^{n_{\text{ts}}} \left\| \boldsymbol{x}_i^{(h)} - \boldsymbol{x}_j^{(h)} \right\|_2, \tag{5.2}$$

where $n_{\text{ts}} = 240$ corresponds to the total amount of control time steps performed in a $60\,\mathrm{s}$ simulation, and $\boldsymbol{x}_i^{(h)}$ indicates the input vector of the i-th voxel at the h-th control time step. In plain words, we introduce a matrix where each element quantifies the average distance between the inputs of each pair of ANNs of a VSR. By observing these matrices for all the considered VSRs, we can draw some conclusions regarding how diverse the inputs of different ANNs are.

We present all the computed matrices in Figure 5.5, organized by controller type (MLP and H-MLP with the two different η values) on the rows and VSRs on the columns. To facilitate visualization and pattern recognition within the matrices, we arrange the voxels based on their *connectivity*, which refers to the presence or absence of neighboring voxels on the four sides (with $2^4 - 1$ different

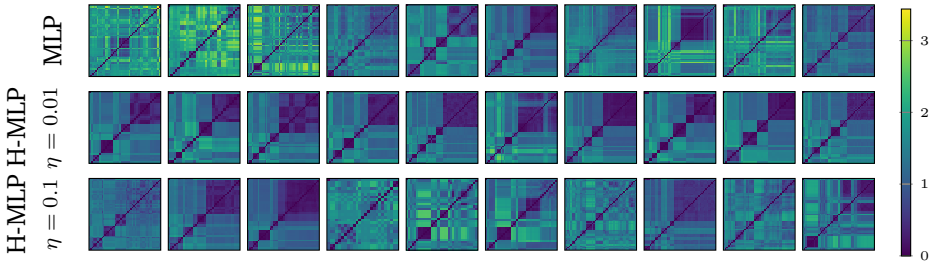


Figure 5.5: Average pairwise distance between inputs observed by the ANNs of each VSR (grouped by controller type on rows, and VSR on columns (seeds 1 to 10), during a simulation of 60 s. The ANNs embedded in different voxels of the same VSR experience different inputs.

values). As a result, each pair of voxels i and $i+1$ considered when computing $\boldsymbol{D}$ either shares the same connectivity type (i.e., they have identical free and connected sides) or usually differs by at most one connected side (e.g., the i-th voxel might have all sides connected while the $(i+1)$-th voxel has one side, such as the top, free).

Given this representation choice, we can observe a distinct checked pattern in the matrices shown in Figure 5.5. This pattern suggests that voxels with similar connectivity tend to have similar experiences. This is likely because voxels with the same connectivity typically play similar roles within the robot: for example, "back" voxels have only the upper side free, while "core" voxels have all sides connected. Square-colored blocks in the matrices indicate that the average distance in the input space is similar for pairs of voxels within the same connectivity families, or roles. Additionally, the darker checks around the diagonals, where distances are computed between voxels of the same type, further confirm that distances are smaller (i.e., more similar) for voxels of the same type.

When comparing the matrices across different VSRs, we observe that some matrices exhibit larger distances (i.e., lighter areas) than others. This variation does not seem to correlate with the learning process, as lighter areas appear in matrices from all three rows. We attribute these differences to the diverse gaits achieved by the VSRs, which could result in varying degrees of differences in the inputs experienced by the voxels.

Based on these results, we can conclude that different voxels experience distinct stimuli during their lifetime. These differences are influenced primarily by two factors: (1) the role of the voxel, expressed by its connectivity, and the (2) the gait achieved by the VSR.

To assess how diverse the learned ANNs become in their responses to identical stimuli, we begin by identifying a set of representative inputs. We achieve this by partitioning the space of collected inputs into $n_k = 25$ clusters using the K-Means clustering algorithm[2]–we also repeat the analysis with different values of n_k finding negligible differences in the final outcomes. We choose the centroids of these clusters as the representative stimuli for further analysis.

Having obtained n_k representative inputs, we employ them to compute the specialization matrix $\boldsymbol{S}$ of all the saved VSRs. We compute the specialization matrix $\boldsymbol{S}$ of a VSR as:

$$\boldsymbol{S} = s_{i,j} = \frac{1}{n_k} \sum_{\boldsymbol{x}_c \in C} ||\text{ANN}_i(\boldsymbol{x}_c) - \text{ANN}_j(\boldsymbol{x}_c)||_2 \,, \tag{5.3}$$

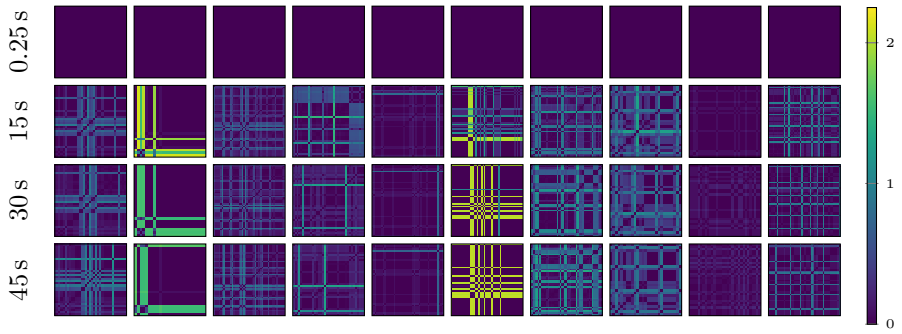
where C indicates the set of n_k previously computed centroids, and ANN_i indicates the ANN pertaining to the i-th voxel. Simply put, each element of $\boldsymbol{S}$ quantifies the average distance in the output space across some representative inputs for the two considered voxels. Clearly, the larger the average distance, the more functionally different the voxels: i.e., for the same inputs, they produce different outputs. As for the input distance matrices $\boldsymbol{D}$, we order the voxels according to their connectivity when computing each $\boldsymbol{S}$, to ease the visual discovery of patterns.

We present the specialization matrices for all the considered VSRs in Figure 5.6. The figure is divided into two sub-figures, each corresponding to a different value of η. We do not compute this matrix for VSRs equipped with MLP-based controllers, as their ANNs are identical by design. In each sub-figure, columns represent different VSRs, while each row displays the ANNs sampled at various control time steps, as indicated on the left of the row.

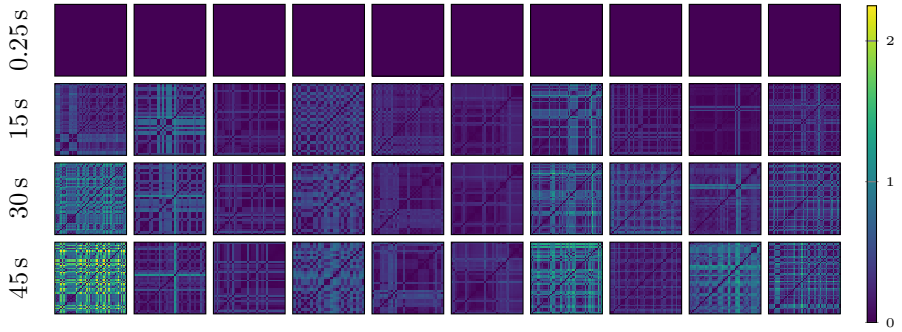
We begin our analysis by examining each column of the figure to understand how the functional differences in the ANNs of a VSR evolve over its lifetime. Initially, all columns show very dark matrices, indicating minimal functional differences. As time progresses, these matrices lighten, revealing increased specialization. This pattern clearly indicates that, despite the ANNs being initialized identically, they develop distinct responses to the same stimuli, leading to functional and behavioral differentiation.

When comparing the various maps, we observe that specialization develops differently across cases. Some maps lighten rapidly, while others remain darker even after $45\,\text{s}$. This supports our earlier conclusion that specialization varies across different individuals. Generally, the matrices in the first sub-figure (for higher η) show lighter colors compared to those in the second sub-figure (for lower η). This suggests that a higher η results in stronger specialization. This

[2] -



(a) H-MLP ($\eta = 0.1$)



(b) H-MLP ($\eta = 0.01$)

Figure 5.6: Specialization matrices $\boldsymbol{S}$. The ANNs embedded in different voxels of the same VSR become functionally and behaviorally different over time.

trend aligns with Equation (2.20), as η controls the intensity of weight adjustments during Hebbian learning.

Examining the final rows of Figure 5.6, we notice the same checked patterns observed in Figure 5.5. This pattern suggests that specialization is closely related to voxel connectivity. However, the patterns here are denser, indicating that there can be functional differences even within the same connectivity family.

To summarize, we confirm that Hebbian learning drives specialization, resulting in behavioral and functional differences among the ANNs. Additionally, we highlight a threefold entanglement between (1) the role and connectivity of a voxel, (2) its experience, and (3) its specialization.

5.5 Concluding Remarks

In this chapter, we examined the concurrent optimization of both the body and brain of VSRs. We employed an ES to optimize both the body configuration (i.e., the arrangement of the modules) and the brain (i.e., the parameters of an ANN within each module). For the ANNs, we integrated Hebbian learning, a form of unsupervised learning where synaptic weights adjust during the agent lifetime based on the signals traversing the synapses.

Our experimental findings revealed that robots utilizing Hebbian learning exhibit superior performance compared to those without this form of plasticity. Specifically, we discovered that: (1) robots with Hebbian learning do learn, i.e., they retain their abilities even if the plasticity is disabled, provided this is done after a long enough learning period; (2) Hebbian learning results in specialization, i.e., modules which are initially identical become different after learning, with their ANNs processing differently the same inputs—in brief, we observed a first form of *totipotency* in artificial organisms and in the ANNs controlling them; (3) specialization is mostly related to the role of the modules, namely, their position, in the overall body of the robot.

6

Scheduling the Development in Morphological Plasticity

Development is essential for living beings. Since robots are often designed to mimic biological organisms, it is believed that development is equally critical for achieving success in robotic agents. However, the optimal timing for this development remains unclear. In biological systems, development primarily occurs during the initial growth phase, but it is not yet known whether this principle applies to artificial creatures as well. In this chapter, we use an evolutionary approach to optimize the development of VSRs, using different representations. In our study, development involves adding new voxels to the VSR at specific time intervals according to the development schedule. *We experiment with different schedules and demonstrate that, similar to living organisms, artificial agents benefit more from early-stage development than from continuous development throughout their lifespan.*

This chapter is based on the following publication: **G. Nadizar**, E. Medvet, and K. Miras. On the Schedule for Morphological Development of Evolved Modular Soft Robots. In *European Conference on Genetic Programming (Part of EvoStar)*, pages 146–161. Springer, 2022 [296].

6.1 Introduction and Related Works

Phenotypic development is widespread in nature and can occur in various dimensions, such as lifetime body adaptations to seasonal changes in the environment [230], brain plasticity through learning [123, 113], physical training [191], and behavioral regulation in response to environmental factors [377], among others. Beyond these forms of development, one of the most fundamental is *growth*. Growth begins during morphogenesis and may continue for a significant portion of the life of an organism, depending on its species. In humans, phenotypic growth fluctuations are influenced by genetic and environmental factors during prepubertal and pubertal stages [414], making it challenging to establish growth standards [53]. Additionally, different body traits develop at various stages; for example, significant height growth occurs until adolescence [245], while male muscle mass typically peaks between the ages of 20 and 30 [26].

Interestingly, body growth in animals is rapid early in life and then gradually slows. The reasons for this pattern are not fully understood [245], though it may be related to the advantages of delaying fertility maturation [178]. Moreover, animal brain development is a lifelong process, occurring prenatally, during infancy, adolescence, and even into adulthood [415]. For instance, the human frontal cortex, due to its complexity, does not fully mature until the mid-twenties [377]. This maturation process involves not just growth but also the reorganization of neural structures, which requires processes such as synaptic pruning [377].

While the dynamics of growth, including the interaction between body and brain development, remain unclear, it is evident that this complex process is crucial for the behavioral complexity observed across species. As such, ER has a strong motivation to study growth development. However, the field has predominantly focused on evolving the controller without considering body evolution [343], and developmental processes have received relatively little attention.

Some developmental models have gained popularity [232, 399], though they have mostly been applied to morphogenesis. One study demonstrated the benefits of environmental regulation for lifetime phenotypic plasticity, allowing robot bodies and brains to adapt to environmental changes [279, 278]. Another approach involved reconfigurable robots with manually designed bodies [80]. Additionally, the field of morphogenetic engineering has been introduced to model complex self-architecting systems [96]. Other studies explored pre-programmed lifetime changes comparable to growth, investigating the impact of development on evolvability [205, 204] and the effects of using stages of morphological development to scaffold behavior [38].

Although these studies represent significant progress in investigating development within artificial life, much remains to be explored. There is a growing need for more research to expand our understanding of how development can be

implemented and under what conditions specific effects may be observed.

To contribute new insights to the literature, this chapter addresses a specific question related to growth development: does the development schedule affect the effectiveness of evolved agents? Specifically, is continuous, lifelong development more or less effective than development that occurs primarily at the beginning of an agent life?

To explore this, we design various developmental models for VSRs [153], which are optimized using appropriate EAs. By combining development with evolution, we enable robots to undergo changes on different timescales. Due to their versatility, VSRs are ideal for experimenting with morphological development, and they have been used in previous studies [204, 450]. However, unlike these studies, which focus on the overall impact of development on evolution [204] and the role of environmental feedback [450], our research specifically investigates the development schedule.

Although our work centers on morphological development, involving only the robot body, the controller is tightly coupled with its morphology. Thus, we also design adaptable brains that can effectively control various body forms. To evaluate the effects of development, we assess robot performance in a locomotion task and include non-developing robots as a baseline. Our results indicate that, across all models, the most effective development schedule for artificial agents mirrors that of living organisms. Specifically, early development leads to better-performing robots than continuous growth. Moreover, the comparison with non-developing robots highlights the potential benefits of development for artificial agents.

6.2 Development of Voxel-based Soft Robots

We consider morphological development, i.e., a mechanism according to which, at given time instants during the life of the VSR, new voxels are added to the VSR body. For the purpose of this study, we say that the development of a VSR is completely described by a schedule and a development function. We define the *schedule* as a sequence $S = (t_j)_j$ of time instants when the addition of a new voxel occurs. We define the *development function* as a function d that, given a number of voxels n, outputs a VSR $d(n)$ consisting of at most n voxels. We impose the requirements for the function d that, for all n, (1) the morphology of the VSR $d(n)$ differs from the morphology of the VSR $d(n+1)$ for at most one voxel and (2) $d(n)$ has no more voxels than $d(n+1)$.

Given a starting size n_0, a schedule S, and a development function d, we can perform a simulation of a VSR $d(n_0)$ that starts with a morphology of (at most) n_0 voxels at $t = 0$ and, at each $t_j \in S$, develops to a VSR $d(n_0 + j)$ that is not

smaller than the previous one.

The primary objective of this study is to understand how the schedule of development affects the effectiveness of evolved VSRs. To explore this, we define a set of predetermined schedules, allow evolution to optimize the development function, and then compare the results.

6.2.1 Representations for the Development Function

To enhance the generality of our experimental findings, we explore four distinct representations of the development function, each designed to be optimized through evolutionary processes.

For the ease of presentation, we describe the development function d in terms of a function d_{morph}, that determines the morphology of the VSR, and a function $d_{\text{controller}}$, that determines the controller of the VSR. Moreover, we directly describe how the outputs of $d_{\text{morph}}(n, g)$ and $d_{\text{controller}}(n, g)$ are computed, given a genotype g and a number n.

Vector-based Morphology Representation

Given a real vector $\boldsymbol{v} \in \mathbb{R}^{n_{\text{side}}^2}$, we obtain a morphology of n voxels as follows. We denote by $\boldsymbol{M} = d_{\text{morph}}(n, \boldsymbol{v})$ the Boolean matrix describing the obtained morphology, where the voxel at position i, j is present if and only if the corresponding element $m_{i,j}$ is set.

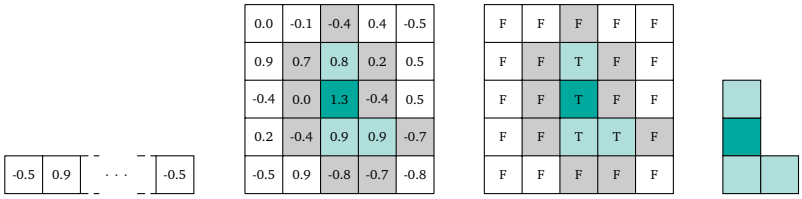
First, we reshape the vector $\boldsymbol{v}$ to a matrix $\boldsymbol{V}$ of n_{side}^2 real values. Second, we determine the greatest element of $\boldsymbol{V}$ and set the corresponding element of $\boldsymbol{M}$. Then, we repeat the following two steps until $\min(n, n_{\text{side}}^2)$ elements of $\boldsymbol{M}$ have been set: (1) we consider the subset of $\boldsymbol{M}$ unset elements that are adjacent to set elements and (2) we set the element of the subset with the largest corresponding value in $\boldsymbol{V}$.

Note that, with this representation, it is guaranteed that the morphology will have exactly n voxels, provided that n_{side}, a parameter of the representation, is large enough.

Figure 6.1 provides a schematic representation of an example of application of this function with $n_{\text{side}} = 5$ and $n = 4$.

Tree-based Morphology Representation

Given an ordered tree T in which each node is a number in $\mathbb{R}$ and has either 0 or 4 child nodes, we obtain a morphology of up to n voxels as follows. Each node of the tree corresponds to an element of the matrix $\boldsymbol{M}$ describing the morphology, while the four children of a node correspond to the four neighboring elements



(a) Real vec. $\boldsymbol{v}$. (b) Real matrix $\boldsymbol{V}$. (c) Bool. mat. $\boldsymbol{M}$. (d) VSR.

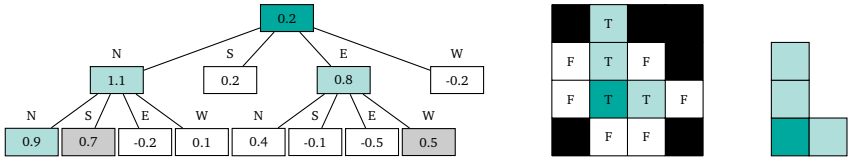
Figure 6.1: Schematic view of the grid-based morphology representation $d_{\text{morph}}(n, \boldsymbol{v})$, with $n_{\text{side}} = 5$, $n = 4$, and an example $\boldsymbol{v} \in \mathbb{R}^{25}$. Dark green is used to highlight the first element chosen (the one with highest value), whereas lighter green is used to indicate the other chosen elements. The gray area indicates the candidate voxels for a possible future development (i.e., $n = 5$ with this same $\boldsymbol{v}$).

at north, south, east, and west. Namely, given a node corresponding to the $m_{i,j}$ element, the first child corresponds to $m_{i,j-1}$, the second to $m_{i,j+1}$, etc.

First, we transform T into a tree T' by mapping each node of T to a node in T'. Nodes in T' are pairs (v, u), where $v \in \mathbb{R}$ is the node real value and $u \in \{\text{SET}, \text{UNSET}, \text{USED}\}$—initially, $u = \text{UNSET}$ for every node in T'. Second, we set the root u to SET. Then, we repeat the following three steps until n nodes in T' have $u = \text{SET}$ or there are no more T' nodes with $u = \text{UNSET}$: (1) we consider the subset of nodes with $u = \text{UNSET}$ and whose parent node has $u = \text{SET}$, (2) we choose the node in the subset with the largest v, and (3) set $u = \text{SET}$ for the chosen node and $u = \text{USED}$ for all the other nodes representing the same position. Finally, we obtain the morphology $\boldsymbol{M}$ by setting the element $m_{0,0}$ first, and proceeding to set every other element i, j for which there is a SET node in T' whose relative position to the root is i, j. For convenience, we assume that the indexes of the elements of $\boldsymbol{M}$ can assume values in $\mathbb{Z}$.

Note that, with this representation, a morphology with less than n voxels could be obtained for a given tree T, depending on its size. In the extreme case, if T consists of the root node only, then the morphology will have just one voxel. On the other hand, the representation is parameter-free and does not impose an upper bound on the number of voxels in the VSR. Note also that, for each position of the matrix $\boldsymbol{M}$ there are up to four nodes in the tree T, but at most one is used depending on ancestors of the node: i.e., this representation is redundant [363] and exhibits epistasis [431].

Figure 6.2 provides a schematic representation of an example of application of this function with $n = 4$.



(a) Tree T'. (b) Bool. matr. $\boldsymbol{M}$. (c) VSR.

Figure 6.2: Schematic view of the tree-based morphology representation $d_{\text{morph}}(n, T)$, with $n = 4$. Colors in T' nodes represent the value of u, while numbers are the ones of T (not shown here for brevity). Dark green is used to highlight the root of the tree, whereas lighter green is for $u = \text{SET}$, white for $u = \text{UNSET}$, and gray for $u = \text{USED}$. The same colors are used in the Boolean matrix $\boldsymbol{M}$ and in the obtained VSR morphology. Black cells in $\boldsymbol{M}$ correspond to nodes that are not present in the tree, hence such cells could never be used with this T, regardless of n.

6

Vector-based Phase Controller Representation

Given a real vector $v \in \mathbb{R}^{n_{\text{side}}^2}$, we obtain a phase controller (see Section 2.1.3) for a VSR whose morphology $\boldsymbol{M}$ can be contained in a $n_{\text{side}} \times n_{\text{side}}$ grid of voxels as follows.

First, we reshape the vector $\boldsymbol{v}$ to a matrix $\boldsymbol{V}$ of $n_{\text{side}} \times n_{\text{side}}$ real values. Second we build a phase controller in which the phase $\phi_{i,j}$ of the voxel at position i, j, if any, is given by the corresponding element in $\boldsymbol{V}$. If there is no voxel at i, j, the corresponding element in $\boldsymbol{V}$ does not contribute to the controller.

Vector-based Neural Controller Representation

Given a real vector $v \in \mathbb{R}^p$ and a description of the topology of an MLP consisting in the numbers of neurons for each layer, we obtain an MLP by simply setting the parameters vector $\boldsymbol{\theta}$ of the MLP to $\boldsymbol{v}$ (see Section 2.2.1). We use the obtained MLP to implement the output function of the distributed controller for the VSR, employing explicit communication among voxels (see Section 2.1.3).

In this case, all output functions of the VSR are the same. Therefore, this controller is applicable to any VSR, regardless of its morphology, provided that (1) the MLP input-layer size is compatible with the dimension $n_{\text{sensor}} = |\boldsymbol{r}_i^{(k)}|$ of sensor readings in the voxels and with the value of n_c, i.e., it is $n_{\text{sensor}} + 4n_c$, and (2) the MLP output-layer size is compatible with the value of n_c, i.e., it is $1 + 4n_c$. Since n_{sensor} is determined by the morphology, it follows that the free parameters for this representations are n_c, m, and the architecture of the MLP.

Tree-based Phase Controller Representation

This controller representation is tightly coupled with the tree-based morphology representation: in fact, we only use it in combination with that representation. We consider an ordered tree T in which each node is a pair of numbers $v, \phi \in \mathbb{R}^2$ and has either 0 or 4 child nodes. From T we map the v-part of T according to the tree-based morphology representation to obtain a VSR with morphology M. We then obtain a phase controller by associating with each voxel in the morphology the ϕ value of the corresponding element in T.

The rationale behind this representation, is to tightly couple v values, which determine the morphology, and ϕ values, which define the controller, by embedding them within the same tree structure. Together with suitable genetic operators, this link should prevent destructive effects resulting from the misalignment between the part of the genotype describing the morphology and the one describing the brain [327].

Full Development Function Representations

Summarizing, we consider the four representations resulting from the following combinations of a d_{morph} and a $d_{\text{controller}}$ representation:

- *Grid-phase*, in which the genotype is a vector $\boldsymbol{v} \in \mathbb{R}^{2n_{\text{side}}^2}$: we obtain the robot by mapping the leading half of $\boldsymbol{v}$ with the vector-based morphology representation and the trailing half with the vector-based phase controller representation.

- *Grid-neural*, in which the genotype is a vector $\boldsymbol{v} \in \mathbb{R}^{n_{\text{side}}^2+p}$: we obtain the robot by mapping the leading n_{side}^2 elements of $\boldsymbol{v}$ with the vector-based morphology representation and the trailing p elements with the vector-based neural controller representation.

- *Tree-phase*, in which the genotype is a tree T with nodes in $\mathbb{R}^2$ with either 0 or 4 children: we obtain the robot by mapping the tree of the first elements of T nodes with the tree-based morphology representation and T with the tree-based phase controller representation.

- *Tree-neural*, in which the genotype is a pair $T, \boldsymbol{v}$ composed of a tree T with nodes in $\mathbb{R}$ with either 0 or 4 children and a vector $\boldsymbol{v} \in \mathbb{R}^p$: we obtain the robot by mapping T with the tree-based morphology representation and $\boldsymbol{v}$ with the vector-based neural controller representation.

We remark that, in principle, we could also employ more sophisticated neural models when considering the distributed neural controller, e.g., RNNs or SNNs.

In addition, we could also enhance the controllers with a form of Hebbian learning, which we have observed to be beneficial in Chapter 5. However, we decide to limit our study to a simple phase controller and to an MLP-based neural controller to avoid combining the effects of multiple techniques which would complicate the analysis of the results.

6.2.2 Evolutionary Optimization of the Development Function

To optimize development functions, we employ a standard Genetic Algorithm (GA) (see Algorithm 2.1) that we adapt, in the initialization and genetic operators, to the four different representations presented above.

For initializing the population, i.e., for the implementation of the `init()` function, we sample $U([-1, 1])$ for each element of the vector-based representations, and we use the ramped half-and-half initialization (with depth in $[d_{\text{min}}, d_{\text{max}}]$) for the tree-based representation. For the latter, we sample $U([-1, 1])$ for the values of the nodes.

Concerning the genetic operators, we implement the `variate()` function by applying a crossover operator with (with probability p_{cross}) or a mutation operator (with probability $1 - p_{\text{cross}}$). In the latter case we only sample one parent with tournament selection.

In the vector-based representations (grid-phase and grid-neural), we use the extended geometric crossover, where the child $\boldsymbol{v} \in \mathbb{R}^p$ is determined from the parents $\boldsymbol{v_1}, \boldsymbol{v_2} \in \mathbb{R}^p$ as $\boldsymbol{v} = \boldsymbol{v_1} + \boldsymbol{\alpha}(\boldsymbol{v_2} - \boldsymbol{v_1}) + \boldsymbol{\beta}$, where $\boldsymbol{\alpha} \in \mathbb{R}^p$ is sampled as $\alpha_i \sim U(-0.5, 1.5)$, and $\boldsymbol{\beta} \in \mathbb{R}^p$ is sampled as $\beta_i \sim N(0, \sigma_{\text{cross}})$. As mutation, we use the Gaussian mutation, where $\boldsymbol{v} = \boldsymbol{v_1} + \boldsymbol{\beta}$, with $\boldsymbol{\beta} \in \mathbb{R}^p$ sampled as $\beta_i \sim N(0, \sigma_{\text{mut}})$.

In the tree-based representations, we use the standard subtree crossover and standard subtree mutation, always ensuring a maximum depth of d_{max} for the child. Only with the tree-phase representation, with $50\,\%$ probability, we apply a noise sampled from $N(0, \sigma_{\text{mut}})$ to each ϕ element of the tree instead of applying the standard subtree mutation.

In the combined representation (tree-neural), we do crossover by applying standard subtree crossover and extended geometric crossover to the two parts of the genotype. Similarly, we do mutation by applying Gaussian mutation and standard subtree mutation.

6.3 Experimental Evaluation

We perform several experiments to answer to the following questions:

1. What is the most appropriate development schedule for artificial agents?

2. Does it depend on the representation of the development function?

For answering to said questions, we evolve development functions to develop VSRs suited for the task of *locomotion* on a flat terrain. We employ two different development schedules, together with no development, to be considered as a baseline. We provide a detailed description of the experimental procedure and of our results in the following.

Concerning the representation, we use the following parameters: $n_{\text{side}} = 10$, $f_{\text{sin}} = 1\,\text{Hz}$, $f_c = f_{\text{sim}}$ (phase controller parameters from Equation (2.1)), $n_c = 2$. For the MLP, we employ two hidden layers with the same size of the input layer and $\tanh$ as activation function. When employing the neural controller, i.e., the MLP, we equip VSRs with area, touch, and vertical and horizontal velocity sensors ($n_{\text{sensor}} = 4$).

Regarding the GA, we use the following parameters: $n_{\text{pop}} = n_{\text{off}} = 96$, $n_{\text{evals}} = 20\,000$, $n_{\text{elite}} = 0$, $p_{\text{cross}} = 0.75$, $n_{\text{tour}} = 10$, $\sigma_{\text{cross}} = 0.1$, $\sigma_{\text{mut}} = 0.35$, $d_{\text{min}} = 3$, and $d_{\text{min}} = 6$. We observe that, for the chosen values of n_{pop} and n_{eval}, evolution is in general capable of converging to a solution, i.e., longer evolutions would result in negligible fitness improvements.

To evaluate the effectiveness of an individual, i.e., a development function, given a schedule S, we proceeded as follows. At the beginning of the simulation, we (1) use the development function to obtain an initial VSR $d(n_0)$ from n_0 and (2) we place it right above the terrain at the starting position. Then, at each $t_i \in S$ during the simulation, we (1) remove the VSR $d(n_{i-1})$ from the simulation, taking note of the x-coordinate x_{left} of its leftmost voxel, (2) use the development function to develop the VSR to $d(n_i)$ and (3) place the developed VSR in the simulation right above the terrain in a position such that its leftmost part is at x_{left}. We stop the simulation after $210\,\text{s}$ (simulated time), take note of the run distance Δx, as the difference between the initial and final x-coordinate of the center of mass of the VSR, and use Δx as fitness.

We remove and add (i.e., *re-spawn*) the VSR just before and right after each development because the new voxel might be added in positions that conflict with the current posture of the robot (e.g., under a foot, "inside" the terrain). As a consequence, each development step leads to a re-spawning of the VSR, where its gait is interrupted.

We characterize the performance of the representations in conjunction with two development schedules, both encompassing 14 stages: *early* development $S_{\text{early}} = (10, 20, 30, \ldots, 120, 130)$, resembling biological development, and *uniform* development $S_{\text{uniform}} = (15, 30, 45 \ldots, 180, 195)$, accounting for continuous growth (numbers are in s). In addition, to have a baseline for comparisons, we also employ a non-developmental schedule $S_{\text{no-devo}} = \emptyset$: in this case VSRs are initialized according to the initial development, and no voxels are ever added to them. To ensure fairness in terms of re-spawning (as such events could slow

down the VSR), we interrupt the gait to lift the VSR in the air according to S_{uniform}, even though no development is occurring.

Concerning the initial size n_0 of the VSRs, we aim at being as fair as possible, since larger robots could, in principle, benefit from having more power. Hence, we choose $n_{0,\text{early}} = 6$, $n_{0,\text{uniform}} = 8$, and $n_{0,\text{no-devo}} = 14$ in order to have approximately the same weighted average VSR size during the simulation ($\bar{n}_{\text{early}} \approx 14.7$, $\bar{n}_{\text{uniform}} = 14.5$, and $\bar{n}_{\text{no-devo}} = 14$). In other words, the chosen values of n_0 result in all VSRs having approximately the same integral of size over the simulations.

For each of the $4 \cdot 3$ combinations of representation and schedule, we perform 10 independent, i.e., based on different random seeds, evolutionary optimizations, obtaining a total of 120 runs. When comparing results of pairs of combinations, we perform the Mann-Whitney U test, after having verified the proper requirements, with the null hypothesis of equality of the means—we report the p-values. Note that, since we perform multiple pairwise comparisons simultaneously, we apply the required Bonferroni corrections for evaluating the significance of results.

We perform the experiments in this chapter using the 2D-VSR-Sim simulator. All the code is publicly available[1].

6.3.1 Results and Discussion

We report our results in Figures 6.3 to 6.5. The most prominent finding of our experiments is illustrated in Figure 6.3, which shows the distributions of the fitness Δx for the best individuals at the end of evolution across different representations and schedules. These plots allow us to compare the outcomes of S_{early} and S_{uniform}. Based on the distributions and the p-values, we can conclude that, generally, early development is not inferior to uniform development, and for phase controllers, it is significantly better. This suggests that development in artificial life resembles that in real life. While organisms that grow continuously tend to become larger, this trait does not necessarily enhance overall performance.

We hypothesize that early development is more effective because the optimization of the agent controller benefits from the brain extended interaction with a fixed body (the final development stage, which is longer than the others). Consequently, we speculate that evolution optimizes the controller for the final body, as being optimal during this extended stage could lead to greater distance covered and thus higher fitness. Conversely, continuous growth seems to impede brain development, as evolution struggles to find an optimal controller that can adapt equally well to the various bodies the controller interacts with.

[1] `https://github.com/giorgia-nadizar/VSREvoDevo`

Comparison Against No Development

Building on Figure 6.3, it is also interesting to compare the results obtained by non-developing VSRs. Based on our earlier assumptions, we would expect these outcomes to be significantly better than those of both early and uniformly developed VSRs, as in this scenario, the brain is optimized for a single body. However, the results presented in the plots are less straightforward, with no clear winner among the non-developed, early-developed, and uniformly-developed VSRs. We attribute these mixed findings to two factors: re-spawning and the absence of development. Regarding re-spawning, it likely slows down the VSRs, even though they are not undergoing development, preventing them from demonstrating a smooth and effective gait. Additionally, we speculate that the lack of actual development may have hindered overall performance, consistent with the findings of Kriegman et al. [204].

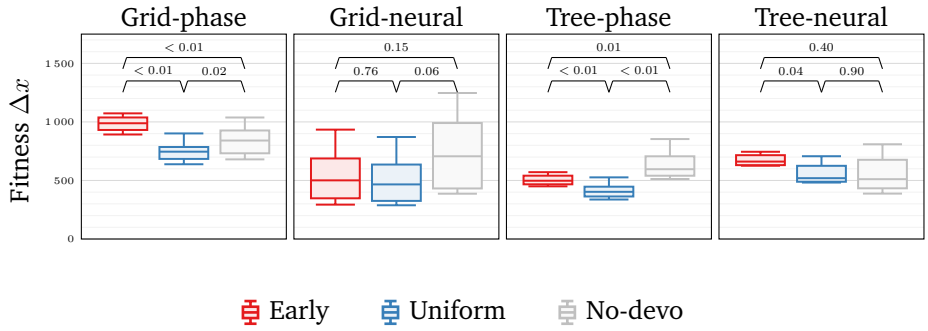


Figure 6.3: Box plots of the fitness Δx of the best development functions at the end of evolution for different representations (plot columns) and development schedules (color). The p-values are shown above pairs of boxes. We consider $\alpha = 0.05/3 \approx 0.017$ for statistical significance, due to the Bonferroni correction.

Analysis of VSR Velocity

To further understand the results, we measure the velocity of VSRs throughout their development, defining $v_{x,i}$ as the average velocity achieved by a VSR during its i-th stage of development. Figure 6.4 shows the distribution of VSRs velocities during the final stage of development for each representation and schedule. These results are consistent with previous findings: early development does not produce slower VSRs compared to uniform development, while non-developed VSRs show less clear relationships with the others. However, more intriguing conclusions emerge when Figure 6.4 is interpreted in light of the different

sizes of VSRs in the final (14th) stage, depending on the development schedule: $n_{14,\text{early}} = 19$, $n_{14,\text{uniform}} = 21$, and $n_{14,\text{no-devo}} = 14$. Interestingly, larger VSRs do not correspond to higher velocities. This can be explained by the same reasoning as before: early development produces VSRs that are primarily optimized for the final stage of their lives, not only due to their larger size but also because the interaction between body and brain has had more time to develop.

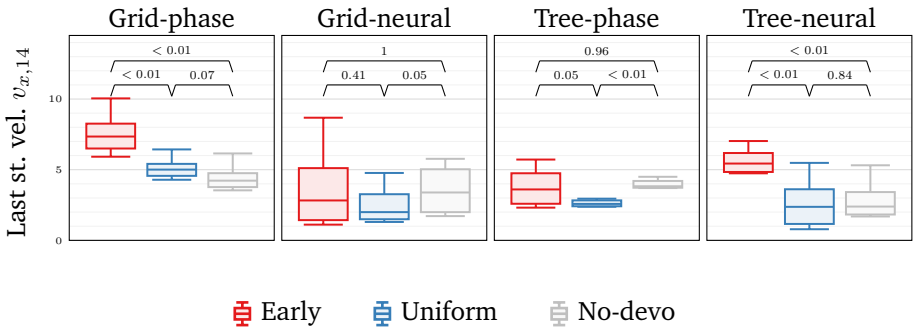


Figure 6.4: Box plots of the last stage velocity $v_{x,14}$ of the best development functions at the end of evolution for different representations (plot columns) and development schedules (color). The p-values are shown above pairs of boxes. We consider $\alpha = 0.05/3 \approx 0.017$ for statistical significance, due to the Bonferroni correction.

To conclude the velocity analysis, we provide in Figure 6.5 a display of the $v_{x,i}$ throughout the simulation. These plots appear to confirm our earlier hypothesis that early development is more successful. Additionally, we observe that both early and uniform development exhibit a general upward trend in velocity, indicating that size also plays a significant role in achieving good locomotion performance.

Comparison Among Representations

Finally, it is worth examining the different outcomes produced by the considered representations, both in terms of fitness Δx (Figure 6.3) and velocity $v_{x,14}$ during the last development stage (Figure 6.4). We summarize the results of statistical significance tests between pairs of representations in Table 6.1, where colored dots (with colors corresponding to the development schedule) mark distributions that are significantly different (p-value $< \alpha = 0.05/6 \approx 0.008$). From the table, it is immediately apparent that the results obtained without development rarely show significant differences based on the representation used. In

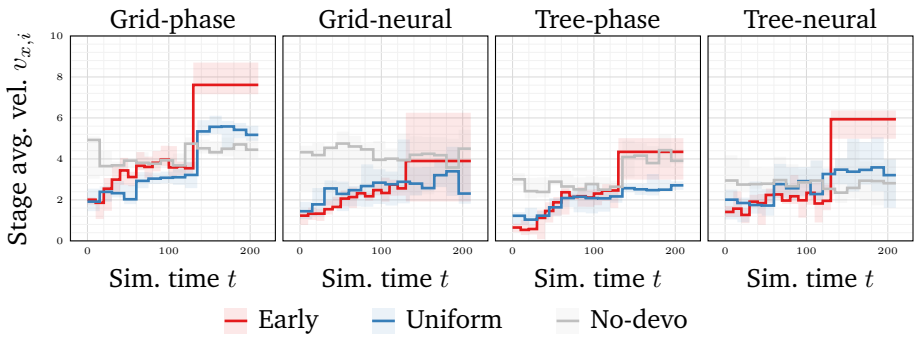


Figure 6.5: Average velocity $v_{x,i}$ (median and inter-quartile range across the 10 repetitions) of the developing VSRs at different stages during the simulation, for different representations (plot columns) and development schedules (color).

contrast, early and uniform development yield more varied outcomes across different representations. Considering Figures 6.3 and 6.4 and Tables 6.1a and 6.1b, we can conclude that the grid-phase representation is generally not worse than the others and is significantly better for certain schedules and representations. We speculate that this could be due to two factors: the direct nature of the representation and the superiority of a phase controller over a neural controller for developing VSRs. Specifically, we hypothesize that it is easier for evolution to determine suitable phase values for a growing body than to optimize a single MLP to accommodate all voxels across a wide range of body types.

	GP	GN	TP	TN
GP	–	•	• • •	• •
GN	•	–		
TP	• • •		–	• •
TN	• •		• •	–

(a) Fitness Δx.

	GP	GN	TP	TN
GP	–	•	• •	• •
GN	•	–		
TP	• •		–	•
TN	• •		•	–

(b) Last stage vel. $v_{x,14}$.

• Early • Uniform • No-devo

Table 6.1: Statistical significance results for different metrics and representations. Each cell is annotated with a dot if the p-value on the two representations with the same schedule is $< \alpha = 0.05/6 \approx 0.008$ (due to the Bonferroni correction).

To conclude the discussion on the experiments we show in Figure 6.6 an example of a developing VSR obtained at the end of one evolution with the grid-

phase representation and the uniform schedule: in the figure, each frame shows the VSR during a developing stage[2].

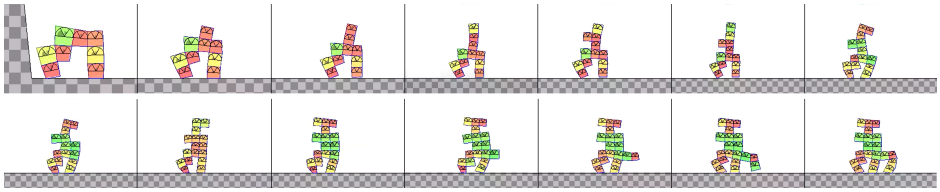


Figure 6.6: View of a developing VSR (uniform schedule with the grid-phase representation). Each image is taken $\approx 0.5\,\mathrm{s}$ after a voxel has been added to the VSR body, to leave time to the robot to fall and exhibit its posture on the ground. Voxels color encodes the ratio between its current area and its rest area (red for contraction, yellow for rest, green for expansion).

6.4 Concluding Remarks

In this chapter, we explored the impact of different morphological development schedules on VSRs. To do so, we evolved development functions—mechanisms that construct and expand VSRs bodies and controllers—to create and develop VSRs capable of successfully performing locomotion tasks. To ensure broad applicability, our study included four representations of development functions based on different combinations of body-brain encodings, along with non-developing VSRs as a baseline for comparison. Our experimental results indicate that, much like in living organisms, VSRs benefit from early development, while continuous growth tends to impede overall performance. Specifically, we observed that larger VSRs resulting from continuous growth, despite having more power, are not more effective than smaller VSRs developed early on. This suggests that an appropriate development schedule is crucial in determining the effectiveness of a VSR.

[2]The corresponding video is available at `https://youtu.be/DD4D20EH1sA`.

7

Synaptic Pruning in Neuroevolution

Robotic agents can be controller with ANNs. These networks are often intricate, with numerous neurons and connections, particularly when the robots are equipped with multiple sensors and actuators spread across their bodies or when a high level of expressive power is required. Pruning, which involves removing neurons or connections, reduces the complexity of ANNs, thereby enhancing their energy efficiency and, in some cases, potentially improving their generalization capabilities. Moreover, pruning in biological neural networks is known to play a crucial role in brain development and learning. In this chapter, we focus on the evolutionary optimization of neural controllers for VSRs applying pruning during their fitness evaluation. We consider the task of locomotion, and experiment with various controllers architectures to assess the impact of different pruning methods on post-pruning effectiveness, long-term effectiveness, adaptability to new terrains, and overall behavior. Our findings indicate that certain forms of pruning in NE produce controllers that are nearly as effective as those evolved without pruning, while also offering greater robustness to pruning. Additionally, we occasionally observe improvements in generalization abilities.

This chapter is based on the following publication: **G. Nadizar**, E. Medvet, O. H. Ramstad, S. Nichele, F. A. Pellegrino, and M. Zullich. Merging pruning and neuroevolution: towards robust and efficient controllers for modular soft robots. *The Knowledge Engineering Review*, 37, 2022 [298].

7.1 Introduction

In recent years, ANNs have been successfully applied to a wide range of problems across various domains. It is well understood that the architecture of an ANN, including the number of neurons, connections (synapses), and other significant hyper-parameters, must be carefully selected to achieve sufficient expressivity for the task at hand. When the ideal network topology is unknown, opting for a large, fully-connected ANN is often considered a safe approach which is often feasible due to the availability of substantial computational power and advanced training algorithms, allowing for the training of very large, potentially over-parameterized networks.

However, the current trend toward scaling neural networks to massive sizes, such as DALL-E with 12 billion parameters [352] or Switch Transformers and Large Language Models (LLMs) with over a trillion parameters [104, 1], has faced criticism due to concerns about carbon footprint and computational costs. Energy consumption and network complexity become particularly critical when ANNs need to be physically implemented in devices with limited resources, such as robotic systems. *Pruning*, which involves removing unnecessary or less important connections to reduce the complexity and energy consumption of ANNs, is an active area of research with a notable biological counterpart. In fact, biological brains undergo a developmental process that initially creates an excess of synapses, which are later optimized through synaptic pruning, a process essential for long-term efficiency [351].

In this chapter, we explore the pruning of ANNs optimized through NE for controlling VSRs. Since each voxel in a VSR may contain sensors, actuators, and the neural controller itself, unnecessary neural connections are undesirable, making pruning a viable strategy. Despite their simplicity, VSRs possess unique characteristics that make them ideal for experimentally studying the effects of real-life phenomena on artificial agents. Moreover, the embodied cognition paradigm, where global behavior emerges from the interaction of simpler behaviors [334], is well-expressed in VSRs, making them particularly suitable for investigating body-brain interactions in artificial agents [234]. Given these attributes, VSRs are excellent candidates for exploring whether synaptic pruning can result in optimized controllers by eliminating redundant or negligible network connections. To address this question, we examine different methods for identifying synapses to prune and experimentally measure the impact of these methods on the overall effectiveness and adaptability of NE-optimized ANNs in controlling VSRs for locomotion tasks.

Our results demonstrate that appropriate pruning strategies applying during evolution can produce controllers that are as effective as those obtained without pruning, while being more robust to pruning in terms of both effectiveness and

behavior. Additionally, we find that individuals evolved with pruning do not show significant reductions in adaptability to new tasks, such as locomotion on unseen terrains, compared to those evolved without pruning.

7.2 Related Works

Our work is related to several topics and lines of research, that are briefly recalled in the next sections. In particular, Section 7.2.1 shows the connections with biological pruning, which occurs during the life of individuals and inspired the adopted pruning scheme. Section 7.2.2 is dedicated to the pruning of ANNs and the various techniques (most of them iterative) that have been recently proposed, especially in the Deep Learning literature. Section 7.2.3 deals with pruning in the context of NE and, finally, Section 7.2.4 briefly recalls the pruning of SNNs, which resemble more closely the biological networks.

7.2.1 Synaptic Pruning in the Nervous System

Biological neural networks are not engineered but rather self-organize, enabling them to adapt and form efficient computational structures [177, 342, 469]. Much of their developmental growth and adaptation hinges on pruning—a process where an initial surplus of neurons, axons, and synapses is followed by the removal of inactive or inefficient components [242, 360, 370]. In humans, this pruning process begins shortly after birth; the neonatal brain, which contains approximately 100 billion neurons, is pruned down to about 86 billion in adulthood, representing a reduction of nearly $15\,\%$. Similarly, the synaptic density between neurons decreases by almost $50\,\%$ in the adult brain compared to that of a 1- to 2-year-old, following an initial postnatal growth [152, 370].

The cellular and molecular mechanisms driving this pruning process are numerous and very complex, but at a higher level, they are thought to be guided by specific constraints, such as metabolic energy efficiency and robustness to perturbation [152, 3, 214, 360]. Biological neural networks must operate within the limits of available local and global metabolic energy, which pushes them to evolve toward more efficient network topologies, leading to the pruning of connections that do not contribute sufficiently relative to their metabolic cost. Conversely, these networks also need to maintain robustness against perturbations like injury or degeneration, necessitating redundancy and adaptability [88]. The interplay of these constraints, sometimes opposing but also complementary, shapes biological neural networks into highly efficient computational structures, often resulting in topologies characterized by small-world, hierarchical, and modular properties [25, 395].

Additionally, these constraints vary across different brain regions, influencing development, including pruning, to create specialized network topologies [396]. Since different brain regions are responsible for distinct computational tasks, the pruning mechanisms adapt to these differences, shaping networks in areas like sensory cortices differently from those involved in executive or motor functions. The network requirements for these tasks—such as redundancy, parallelization, recurrency, and interregional signaling—demand varied pruning targets and timescales [41, 229, 271, 378, 438]. Despite this, the underlying pruning mechanisms are consistent across regions. Generally, neurons are pruned based on their activity levels: low-activity neurons or synapses are marked for removal either autonomously or by microglia (glial immune cells) [13, 360, 378]. The input to each region influences the activity of its neurons, with those contributing more frequently to the output being preserved, enabling the region to adapt to its input while retaining the most efficient network components. By initially growing a large network and then pruning it to match specific computational tasks, the human nervous system is better able to adapt to diverse environments compared to constructed or engineered networks.

These biological insights prompt questions about the design of ANN controllers for artificial agents, particularly regarding the appropriate network size and the number of parameters required to learn a specific task. The potential for optimizing these networks in terms of learning efficiency, robustness to noise, and factors like energy cost remains an active area of research. In the following sections, we review various pruning strategies for ANNs.

7.2.2 Pruning in ANNs

The term “pruning” is not exclusive to ANNs; it originated in the context of decision trees as early as 1984 [45]. In this context, it refers to methods for structural simplification, such as the “pruning” of branches in large trees that tend to overfit or poorly generalize to unseen data. Since its inception, pruning—essentially the top-down removal of excess components from a machine learning model—has been motivated by the pursuit of simplicity, echoing the principle of Occam’s razor [413, 474].

In the context of ANNs, pruning involves various techniques aimed at *sparsifying* the network—i.e., removing connections (synapses) between neurons to create a more streamlined subnetwork from the original model. The reasons for pruning can vary: some techniques may reduce the model error [217, 143], enhance training and inference efficiency, improve generalization [21], or increase robustness [467]. Additionally, pruning can be used to explore similarities between artificial and biological sparsification processes, as discussed in Section 7.2.1.

ANN pruning can be categorized into two main types: *structured* and *unstructured*, based on the group(s) of connection(s) targeted for removal [11]. Structured pruning focuses on removing well-defined groups of synapses, such as *all* connections to a particular neuron or specific channels in Convolutional Neural Networks (CNNs). This approach offers immediate computational benefits, as eliminating entire neurons or filters results in smaller parameter tensors and faster calculations. Unstructured pruning, on the other hand, removes individual connections without regard for their arrangement. This results in irregular sparsity that does not directly affect how parameter tensors are stored in memory. To fully utilize the reduced number of connections, specialized software (like CUSPARSE [307]) or hardware (such as GPUs with dedicated sparsity support) is required [238]. Despite this, unstructured pruning often enhances model performance even at high pruning rates [118, 357], making it a powerful regularization technique. In contrast, models pruned using structured techniques generally have more difficulty matching the performance of unpruned networks, though recent advances have begun to address this issue [54]. Notably, structured pruning can also serve as a regularizer without necessarily removing the pruned parameters from the network structure.

Another way to categorize ANN pruning techniques is based on the heuristics used for connection removal. These heuristics fall into two main types [156]: (1) data-free heuristics, which prune synapses solely based on the state of the network parameters, and (2) data-driven heuristics, which involve evaluating the model on a given data batch to guide pruning decisions. The key difference between these heuristics is that data-free methods enable rapid pruning by focusing on the parameters alone, while data-driven techniques allow for a more nuanced selection process by considering additional criteria such as information flow within the network [412], gradient flow, and Hessian values [217].

In this work, we will employ both types of heuristics for pruning. Specifically, we will use Least-Magnitude Pruning (LMP) [34, 143], a data-free technique that removes connections with small magnitudes, and variants of Contribution Variance Pruning (CVP) [412], a data-driven heuristic that eliminates parameters with low variance (potentially reintegrating the average into the bias term of the corresponding layer). Additionally, we will apply another data-driven heuristic based on the value or magnitude of the signal passing through each synapse. To establish a baseline, we will also use random pruning as a control technique for comparison with the other pruning methods. We provide a detailed overview of these techniques in Section 7.3.

These pruning techniques are commonly applied within the framework of gradient-based ANN training methods, e.g., Stochastic Gradient Descent (SGD). When improving the network through iterative training, we can distinguish between in-training sparsification techniques and after-training techniques [156].

In-training pruning techniques are applied during the training process. A notable example is LASSO, initially developed for linear models [376, 420], which uses an L1-norm penalty term. This concept has been adapted to ANNs [30]. More recent approaches [231] involve feedback mechanisms to reactivate connections that were prematurely removed.

After-training pruning techniques, which include both structured and unstructured methods like LMP, are applied once the network has been fully trained. The immediate effect of pruning can be a loss of performance, which must be restored through re-training of the sparser model [217]. This often leads to an iterative process of training, pruning, re-training, and re-pruning. Different methods vary in their re-training schedules, and there is no clear consensus on which approach yields the best results. For example, there is debate about whether full re-training [118, 357], fine-tuning [238], or hybrid methods [484, 468] achieve higher accuracy. Additionally, the impact of pruning and re-training on the features learned by pruned models is still under investigation [9, 10]. From a computational perspective, in-training pruning is advantageous because it requires only a single training pass. However, after-training methods often achieve higher performance at high levels of sparsity. Recent research by Liu et al. [237] has significantly narrowed this gap by drawing inspiration from biological pruning processes, specifically neuroreconstruction—the ability of biological networks to reconstruct previously removed synapses. This work demonstrates that in-training pruning can achieve performance close to that of dense ANNs. It also introduces a new state-of-the-art for sparse-to-sparse training, which involves training pruned ANNs with randomly re-initialized parameters. Previously, methods relied on either: (1) re-training a pruned ANN with the same parameters as before, or (2) re-training a pruned ANN with parameters reverted to their initial values before the network was trained.

7.2.3 Pruning ANNs in the Context of Neuroevolution

Iterative pruning is challenging to apply in NE, because it does not use an iterative training process like SGD. Instead, NE adjusts the parameters and structure of the ANN using evolutionary variation operators. There is no traditional training phase; instead, ANNs undergo random variations in each generation. For example, NEAT [401], a key method in NE, employs both crossover and mutation to facilitate structural growth and weight modification. Typically, when evolving an ANN with NEAT, the network starts off small and expands as new generations are produced. This approach contrasts with the common Deep Learning paradigm, which usually begins with a very large, overparameterized ANN, as larger models are known for their superior generalization capabilities [309], particularly in fields like Natural Language Processing (NLP) [48].

This does not imply that pruning cannot be integrated into the evolutionary process. For example, Real et al. [354] included a parameter removal phase (equivalent to pruning) in their evolutionary algorithm. Additionally, Evolutionary Acquisition of Neural Topologies (EANT) [183], a variant of NEAT, incorporates pruning as a method for structural modification of the ANN.

Pruning techniques do not need to be tied to a NE algorithm, as they can be independent of the training or evolutionary method and decoupled from it, as we suggest in this work. For instance, Siebel et al. [388] applied pruning to neural controllers using a technique inspired by [217]. More recently, Gerum et al. [126] used random pruning on neural controllers and found that it improved generalization when these controllers were used to navigate agents through a maze. While this study is perhaps the most similar to ours, our findings differ, as we observe that random pruning has a negative impact in our experiments.

7.2.4 Pruning Biologically-Inspired ANNs

SNNs are a type of ANN that is more closely aligned with biological neural processes compared to traditional ANNs. Building on the insights into human brain connectivity discussed in Section 7.2.1, several studies have explored pruning in SNNs. For instance, Iglesias et al. [167] used a criterion similar to CVP to prune SNNs and analyze the connectivity patterns across various pruning iterations. Additionally, Shi et al. [387] implemented LMP on SNNs during training but did not manage to restore the performance of the original, unpruned networks.

7.3 Pruning Techniques

In this work we consider MLPs as ANNs, and consider different forms of pruning. We limit our focus to MLPs for two main reasons: (1) most works in the literature consider this type of ANN, as presented in Section 7.2, and (2) more complex types of ANNs, though suitable for pruning—e.g., RNNs [127], SNNs [167], or plastic MLPs [138]—tend to compensate for other factors (as seen in Chapters 3 to 5), which would make the interpretation of results more complex.

Concerning the employed forms of pruning, they share a common working scheme and differ in three parameters that define an instance of the scheme: the *scope*, i.e., the subset of connections that are considered for the pruning, the *criterion*, defining how those connections are sorted in order to decide which ones are to be pruned first, and the *pruning rate*, i.e., the rate of connections in the scope that are actually pruned. In all cases, the pruning of a connection corresponds to setting to 0 the value of the corresponding element θ_i of the network parameters vector $\boldsymbol{\theta}$.

Since we are interested in the effects of pruning of ANNs used as controllers for robotic agents, we assume that the pruning can occur during the life of the agent, at a given time t_p. As a consequence, we may use information related to the working of the network up to the pruning time, as, e.g., the actual values computed by the neurons, when defining a criterion.

Algorithm 7.1 shows the general scheme for pruning. Given the vector $\boldsymbol{\theta}$ of the parameters of the ANN, we first partition its elements, i.e., the connections between neurons, using the scope parameter (as detailed below): in Algorithm 7.1, the outcome of the partitioning is a list $(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n)$ of lists of indices of $\boldsymbol{\theta}$. Then, for each partition, we sort its elements according to the criterion, storing the result in a list of indices $\boldsymbol{h}$. Finally, we set to 0 the $\boldsymbol{\theta}$ elements corresponding to an initial portion of $\boldsymbol{h}$: the size of the portion depends on the pruning rate ρ and is $\lfloor |\rho \boldsymbol{h}| \rfloor$.

```
1  function prune(θ):
2      (h_1, ..., h_n) ← partition(θ, scope)
3      foreach j ∈ {1, ..., n} do
4          h ← sort(h_j, criterion)
5          foreach k ∈ {1, ..., ⌊|ρh|⌋} do
6              θ_{h_k} ← 0
7          end
8      end
9      return θ
10 end
```

Algorithm 7.1: The algorithm for pruning a vector $\boldsymbol{\theta}$ of ANN parameters given the parameters scope, criterion, and pruning rate ρ.

We explore three options for the scope parameter and five for the criterion parameter; concerning the pruning rate $\rho \in [0, 1]$, we experiment with many values (see Section 7.4).

For the scope, we have:

- *Network*: all the connections are put in the same partition.
- *Layer*: connections are partitioned according to the layer of the destination neuron.
- *Neuron*: connections are partitioned according to the destination neuron (also called post-synaptic neuron).

For the criterion, we have:

- *Weight*: connections are sorted according to the absolute value of the corresponding weight. This corresponds to LMP (see Section 7.2).

- *Signal mean*: connections are sorted according to the mean value of the signal they carried from the beginning of the life of the robot to the pruning time.
- *Absolute signal mean*: similar to the previous case, but considering the mean of the absolute value.
- *Signal variance*: similar to the previous case, but considering the variance of the signal. This corresponds to CVP (see Section 7.2).
- *Random*: connections are sorted randomly.

All criteria work with ascending ordering: lowest values are pruned first. Obviously, the ordering does not matter for the random criterion. When we use the signal variance criterion and prune a connection, we take care to adjust the weight corresponding to the bias of the neuron the pruned connection goes to by adding the signal mean of the pruned connection: this basically corresponds to making that connection carry a constant signal.

We highlight that the three criteria based on signal are data-driven; on the contrary, the weight and the random criteria are data-free. In other words, signal-based criteria operate based on the experience the ANN acquired up to the pruning time. As a consequence, they constitute a form of adaptation acting on the time scale of the robot life, that is shorter than the adaptation that occurs at the evolutionary time scale; that is, they are a form of *learning*. As such, we might expect that, on a given robot that acquires different experiences during the initial stage of its life the pruning may result in different outcomes—similarly to what we have observed for Hebbian learning and specialization in Chapter 5. Conversely, the weight criterion always results in the same outcome, given the same robot. In principle, hence, signal-based criteria might result in a robot being able to adapt and perform well also in conditions that are different than those used for the evolution.

7.4 Experimental Evaluation

We perform various experiments to the extent of answering to the following questions:

1. Is the evolution of effective VSR controllers hindered by pruning? What are the factors that mostly influence the effects of pruning?
2. Does pruning have an impact on the adaptability of the evolved VSR controllers to different tasks? Is the impact of pruning dependent on the same factors highlighted before?

3. Can evolution find a path towards VSR controllers that are *life-long* effective, i.e., effective both before and after pruning? How do these controllers perform in terms of adaptability?

For answering these questions, we experiment evolving the controller parameters of various combinations of controller architectures, ANN topologies, and VSR morphologies. During the evolution, we enable different variants of pruning, including, as a baseline, the case of no pruning. We consider the task of locomotion, in which the goal for the robot is to travel as fast as possible on a terrain. We describe in detail the experimental procedure and discuss the results in Section 7.4.2. We re-evaluate each evolved VSR on different terrains to measure its adaptability, as described in Section 7.4.3. In order to evaluate whether a VSR can evolve to be effective both with and without pruning, we repeat the experimental procedure presented before, with some minor variations, thoroughly detailed in Section 7.4.4.

For evolved VSRs, we also examine the resulting behaviors, performing a systematic analysis based on the features proposed by Medvet et al. [263], which should capture the different gaits achieved by VSRs. We provide a brief description of the analysis pipeline and of the aforementioned features together with the obtained results in Section 7.4.5.

In order to reduce the number of variants of pruning to consider when answering to the aforementioned questions, we first perform a set of experiments to assess the impact of pruning in a static context, i.e., in MLPs not subjected to evolutionary optimization and not used to actually control a VSR. We refer to these conditions as static and disembodied and present the experiments and the corresponding findings in the next section.

7.4.1 Characterization of Pruning Variants in Static and Disembodied Conditions

First, we aim at evaluating the effect of different forms of pruning on ANNs in terms of how the output changes with respect to no pruning, given the same input. In order to make this evaluation significant with respect to the use case of this study, i.e., ANNs employed as controllers for VSRs, we consider ANNs with topologies that resemble the ones used in the next experiments and feed them with inputs that resemble the readings of the sensors of a VSR doing locomotion.

For the ANN topology we consider three input sizes $m_0 \in \{10, 25, 50\}$ and three depths $l^\star \in \{0, 1, 2\}$, resulting in $3 \cdot 3 = 9$ topologies, all with a single output neuron (see Section 2.2.1 for more context on parameters). For the topologies with hidden layers, we set the hidden layer size to the size of the input layer. In all cases we employ $\tanh$ as activation function. In terms of the dimensionality p of the vector $\boldsymbol{\theta}$ of the parameters of the ANN, the consider ANN topologies

correspond to values ranging from $p = (10 + 1) \cdot 1 = 11$, for $m_0 = 10$ and $l^\star = 0$, to $p = (50 + 1) \cdot (50 + 1) \cdot (50 + 1) \cdot 1 = 132\,651$, for $m_0 = 50$ and $l^\star = 2$, where the $+1$ is the bias. We instantiate 10 ANNs for each topology, setting $\boldsymbol{\theta}$ by sampling the multivariate uniform distribution $U([-1, 1])^p$ of appropriate size, hence obtaining 90 ANNs.

Concerning the input, we feed the network with sinusoidal signals with different frequencies for each input, discretized in time with a time step of $\Delta t = 1/10\,\mathrm{s}$. Precisely, at each time step h, with $t = h\Delta t$, we set the ANN input to $\boldsymbol{x}^{(h)}$, with $x_i^{(h)} = \sin\left(\frac{h\Delta t}{i+1}\right)$, and we read the single output $y^{(h)} = f_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(h)}\right)$.

We consider the $3 \cdot 5$ pruning variants (scope and criteria) and 20 values for the pruning rate ρ, evenly distributed in $[0, 0.75]$. We take each one of the 90 ANNs and each one of the 300 pruning variants, we apply the periodic input for $10\,\mathrm{s}$, triggering the actual pruning at $t_p = 5\,\mathrm{s}$, and we measure the mean absolute difference e between the output $f_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(h)}\right)$ during the last $5\,\mathrm{s}$, i.e., after pruning, and the output $f_{\hat{\boldsymbol{\theta}}}\left(\boldsymbol{x}^{(h)}\right)$ of the corresponding unpruned ANN:

$$e = \frac{1}{50} \sum_{h=50}^{h=100} \left\| f_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(h)}\right) - f_{\hat{\boldsymbol{\theta}}}\left(\boldsymbol{x}^{(h)}\right) \right\|. \tag{7.1}$$

Figure 7.1 summarizes the outcome of this experiment. It displays one plot for each ANN topology (i.e., combination of $l^\star$ and m_0) and one line showing the mean absolute difference e, averaged across the 10 ANNs with that topology, vs. the pruning rate ρ for each pruning variant: the color of the line represents the criterion, the line type represents the context. Larger ANNs are shown in the bottom right of the matrix of plots.

From Figure 7.1 we can make the following observations. First, the criterion used for pruning (indicated by the color of the line in Figure 7.1) has the most significant impact on the output of the pruned ANN. Criteria based on weight and absolute signal mean consistently yield lower values for the error difference e, regardless of the scope and pruning rate. Conversely, using the signal mean criterion results in larger values of e even at low pruning rates; beyond $\rho > 0.1$, e does not seem to increase further. Interestingly, the random criterion generally has a less negative impact on e compared to the signal mean criterion. This result can be attributed to the nature of the input provided to the ANNs, which consist of sinusoidal signals. Since the mean of periodic signals with a period shorter than the time before pruning is close to 0, connections that carry relevant information tend to be pruned. We choose sinusoidal signals for this study because they simulate the sensor readings a VSR might collect during locomotion, especially when performing repetitive gaits, which are characterized by repeated movements over time.

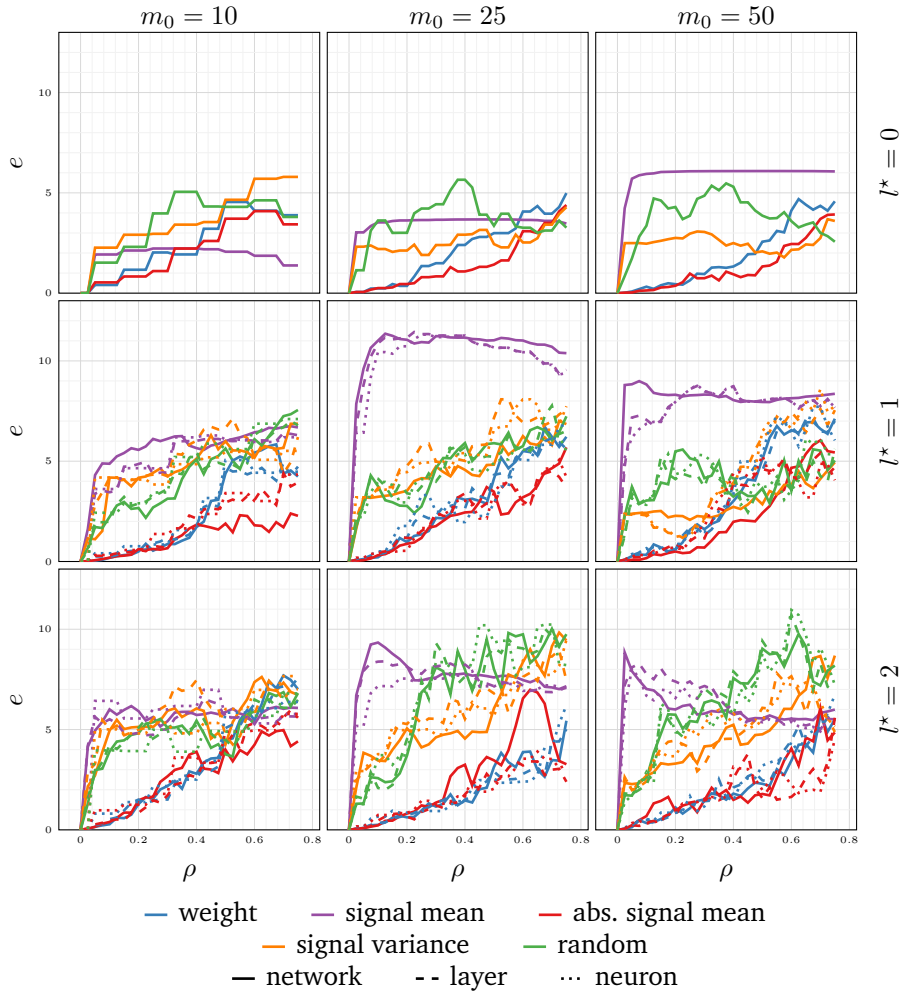


Figure 7.1: Mean absolute difference e between the output of a pruned ANN and the output of the corresponding unpruned ANN vs. the pruning rate ρ, for different ANN structures and with different pruning criteria (color) and scopes (linetype).

Second, there appear to be no significant differences among the three values for the scope parameter. As anticipated, for shallow ANNs (with $l^\star = 0$), the scope parameter has no effect since there is only one layer and a single output neuron, which serves as the destination for all connections.

Third, the pruning rate ρ affects e as expected: generally, a higher ρ leads to a larger e. However, the relation between e and ρ varies depending on the pruning criterion. For the weight and absolute signal mean criteria, Figure 7.1 indicates a linear relation. In contrast, for other criteria, e increases rapidly with ρ and then stabilizes (as seen with the signal mean criterion) or increases more slowly (as observed with the signal variance and random criteria).

Fourth and finally, the topology of the ANN seems to have a minimal effect on the impact of pruning. The depth of the ANN ($l^\star$) appears to have only a slight influence on the differences among pruning variants: deeper networks tend to show less clear distinctions. Regarding the number of inputs (m_0), Figure 7.1 does not provide sufficient evidence to draw any strong conclusions.

Based on the results of this experiment, summarized in Figure 7.1, we decide to consider only weight, absolute signal mean, and random criteria and only the network scope for the next experiments.

To better understand the actual impact of the chosen pruning variants on the output $y^{(h)}$ of an ANN, we show in Figure 7.2 the case of two ANNs. The figure shows the value of the output of the unpruned ANN (in gray), when fed with the input described above (up to $t = 20\,\mathrm{s}$), and the outputs of the $3 \cdot 4$ pruned versions of the same ANN, according to the three chosen criteria and four values of ρ.

7.4.2 Impact on the Evolution

In order to understand if the evolution of VSR controllers is hindered by pruning, we perform various experiments. First, we evaluate the effect of pruning on different controller architectures and ANN topologies. To this extent, we evolve nine VSR controllers, resulting from the combination of three architectures and three ANN topologies, and one morphology.

We experiment with the following controller architectures, among those presented in Section 2.1.3. First, we consider the centralized controller. Then, we consider two variants of the distributed controller with explicit communication, which we call homo-distributed and hetero-distributed. In the former, the ANNs of all voxels are the same, whereas in the latter each voxel has its own ANN. In all cases, since we consider stateless ANNs, the MLP implements the output function of the controller. Concerning the parameters, for the centralized controller and the homo-distributed one, the vector $\boldsymbol{\theta}$ corresponds to the vector of the weights of the MLP, whereas for the hetero-distributed controller, $\boldsymbol{\theta}$ is the concatenation

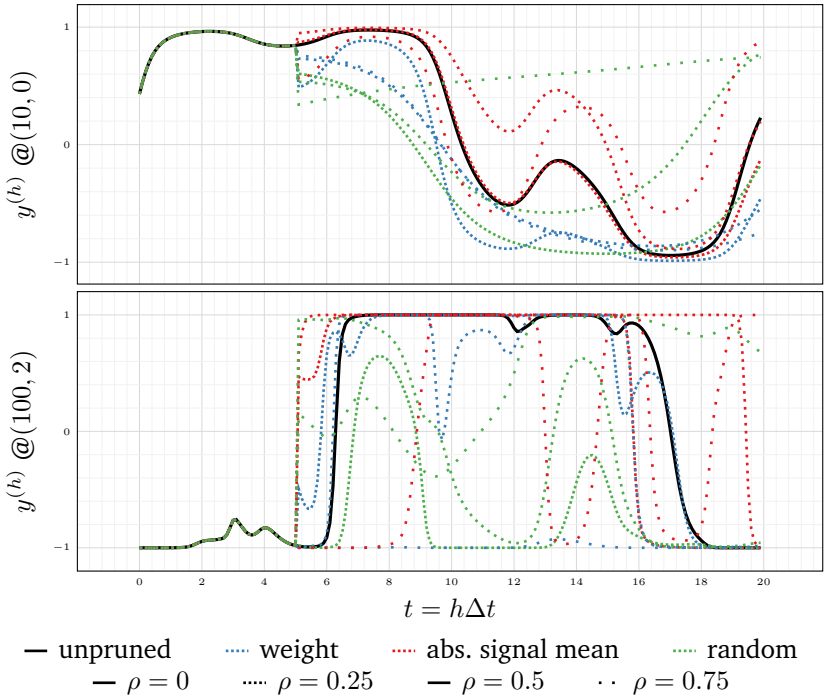


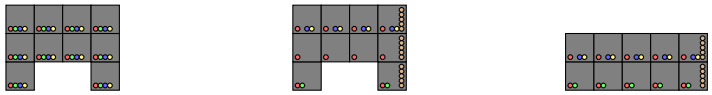
Figure 7.2: Comparison of the output of pruned and unpruned versions of two ANNs of different structures: $m_0 = 10$, $l^\star = 0$, above, and $m_0 = 100$, $l^\star = 2$, below. Pruning occurs at $t_p = 5\,\mathrm{s}$.

of the parameters of each MLP, $\boldsymbol{\theta} = [\boldsymbol{\theta_1}\ \boldsymbol{\theta_2} \ldots \boldsymbol{\theta_n}]$—$n$ being the number of voxels of the VSR.

We combine each of these with different ANN topologies, considering ANNs with $l^\star \in \{0, 1, 2\}$. For the ANNs with hidden layers, we set the size of those layers to match the size of the input layer. Regarding the distributed controllers, we set $n_c = 2$, and for the hetero-distributed architecture we keep the ANN topology homogeneous throughout the entire VSR. In all cases we employ $\tanh$ as activation function in the ANNs.

Concerning the VSR morphology, we employ the biped, , which consists of 10 voxels arranged in a 4×3 grid. We experiment with two different sensor configurations: *uniform*, where each voxel is equipped with velocity (both vertical and horizontal), touch, and area sensors (shown in Figure 7.3a); and *rich*, with area sensors in each voxel, velocity sensors (both vertical and horizontal) in the voxels in the top row, touch sensors in the voxels in the bottom row, and proximity

sensors (with directions $-\frac{1}{4}\pi$, $-\frac{1}{8}\pi$, 0, $\frac{1}{8}\pi$, $\frac{1}{4}\pi$ with respect to the voxel positive x-axis) in the voxels of the rightmost column (shown in Figure 7.3b). These two configurations result in 40 and 35 overall sensor readings, respectively.



(a) Biped, uniform (b) Biped, rich (c) Worm, rich

area touch x-velocity y-velocity proximity

Figure 7.3: VSR morphologies and sensory apparatuses.

We combine the rich configuration with the centralized and with the hetero-distributed controller architectures, whereas we use the uniform configuration in conjunction with the homo-distributed architecture due to its requirements of having the same amount of sensors in each voxel. Table 7.1 summarizes the number of parameters to be optimized for each VSR controller we evolve.

$l^\star$	Centralized	Hetero-dist.	Homo-dist.
0	360	1125	117
1	1620	2623	273
2	2880	4121	429

Table 7.1: Number of parameters to be optimized for each controller architecture and ANN topology.

For each of the nine combinations of controller architecture and ANN topology, we use three different pruning criteria: weight, absolute signal mean, and random, all with network scope, as thoroughly described in Section 7.3. For each criterion, we employ the following pruning rates: $\rho \in \{0.125, 0.25, 0.5, 0.75\}$. We remark that for distributed controllers we apply pruning separately for each voxel ANN. Furthermore, we evolve, for each combination, a controller without pruning to have a baseline for meaningful comparisons.

To perform evolution we use the ES of Algorithm 2.2 with the following parameters: $n_{\text{pop}} = 48$, $n_{\text{evals}} = 20\,000$, $n_{\text{elite}} = n_{\text{pop}}/4 = 12$, and $\sigma = 0.35$. We optimize VSRs for the task of *locomotion*: the goal of the VSR is to travel as fast as possible on a terrain along the positive x-axis. We quantify the degree of achievement of the locomotion task of a VSR by performing a simulation of duration t_f and measuring the VSR average velocity v_x, discarding the initial transitory of duration t_i. In the ES we hence use v_x as fitness for selecting the

best individuals. We set $t_f = 60\,\mathrm{s}$ and $t_i = 20\,\mathrm{s}$ to discard the initial transitory phase. For the controllers with pruning, we set the pruning time at $t_p = 20\,\mathrm{s}$: this way the evaluation of the fitness of the VSR only takes into consideration the velocity after pruning. In Section 7.4.4, instead, we investigate the effects of determining the VSR fitness considering both the pre- and the post-pruning velocities ($t_i < t_p$).

For favoring generalization, we evaluate each VSR on a different randomly generated hilly terrain, i.e., a terrain with hills of variable heights and distances between each other. To avoid propagating VSRs that were fortunate in the random generation of the terrain, we re-evaluate, on a new terrain, the fittest individual of each generation before moving it to the population of the next generation.

For each of the $3 \cdot 3 \cdot (3 \cdot 4 + 1)$ combinations of controller architecture, ANN topology, pruning criterion, and pruning rate (the $+1$ being associated with no pruning) we perform 10 independent, i.e., based on different random seeds, evolutionary optimizations of the controller. We hence perform a total of 1170 evolutionary optimizations. We use 2D-VSR-Sim [261] for the simulation, setting all parameters to default values.

Impact of the Controller Architecture and the ANN Topology

Figure 7.4 summarizes the findings of this experiment. In particular, the plots show how the pruning rate ρ impacts the fitness of the best individual of the last generation, for the different controller architectures and ANN topologies employed in the experiment.

The most striking aspect of the plots is that individuals whose controllers were pruned using the weight or absolute signal mean criteria significantly outperform those subjected to random pruning. This finding indicates that random pruning during each fitness evaluation is detrimental to the evolutionary process. In fact, individuals with strong genotypes might perform poorly if critical connections are removed, while others could excel due to a fortuitous pruning, potentially distorting the selection of the fittest individuals for survival and reproduction. Furthermore, Figure 7.4 confirms that the heuristics based on weight and absolute signal mean criteria (see Section 7.3) effectively identify and remove less crucial connections, thereby minimizing the negative impact of pruning.

Additionally, when comparing the subplots in Figure 7.4, there are no significant differences between the rows. This suggests that the architecture of the controller does not play a crucial role in determining the impact of pruning on its performance.

Similarly, different ANN topologies do not appear to be affected differently by pruning, as there are no clear distinctions between the columns of the plots. However, the first subplot does stand out, as the trend of the lines suggests that

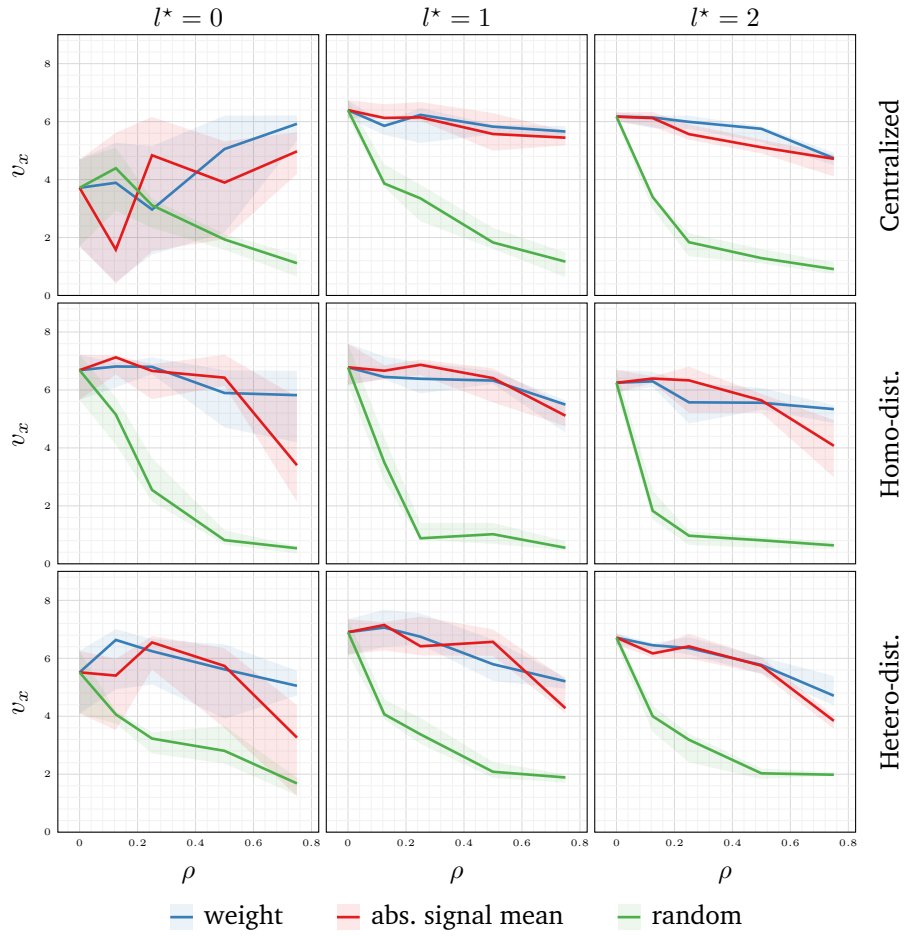


Figure 7.4: Fitness v_x of the best individual (median and inter-quartile range across the 10 repetitions) vs. pruning rate ρ, for different pruning criteria (color), controller architectures (plot row), and ANN topologies (plot column).

pruning might have a beneficial effect on the centralized controller architecture with no hidden layers. Despite this, the wide inter-quartile range indicates that the distribution of fitness v_x is spread over a large interval, making it difficult to draw any definitive conclusions about the potential benefits of pruning for this controller.

For all other subplots, we observe that a higher pruning rate ρ generally leads to weaker controller performance. This outcome aligns with expectations, as increasing ρ means more connections are removed from the ANN, thereby reducing its expressiveness. Nevertheless, controllers pruned using an appropriate heuristic have evolved to achieve results comparable to those that were not pruned during their evolution, which serves as the baseline. To further analyze this, we perform a Mann-Whitney U test with the null hypothesis that, for each combination of controller architecture, ANN topology, pruning criterion, and pruning rate ρ, the distribution of the best fitness is the same as that of the corresponding baseline controller. The baseline refers to a controller with the same architecture and ANN topology but evolved without pruning. The test results show that the p-value is greater than 0.05 in 66 out of 108 cases, suggesting that in these instances, pruning does not significantly affect the performance compared to the baseline.

Impact of the VSR Morphology

Given the lack of significant differences between the centralized and distributed controller architectures, we decide to focus on evaluating the impact of pruning on various VSR morphologies using only the centralized controller architecture. To achieve this, we perform the evolutionary optimization of three additional VSRs, combining each of the three ANN topologies used in the previous experiment with the worm morphology.

Such morphology consists of 10 voxels arranged in a 5×2 grid, ▦. We equip the VSR with the rich sensor configuration, similarly to what we have done in the previous experiment with the centralized controller architecture and the biped morphology (see Figure 7.3c). The amount of parameters to be optimized is the same as in the first column of Table 7.1, as the two VSR morphologies share the same amount of voxels and sensor readings, hence the number of inputs and outputs of the controller ANNs is the same in both cases.

We repeat the exact experimental pipeline as before, employing the same pruning criteria and pruning rates, without changing any hyper-parameter. For each of the $3 \cdot (3 \cdot 4 + 1)$ combinations of ANN topology, pruning criterion, and pruning rate, we perform 10 independent evolutionary optimizations, for a total of 390 runs.

The results are displayed in Figure 7.5, together with the outcomes of the previous experiment for the centralized controller architecture for the biped mor-

phology. The conclusions we can draw from this matrix of plots are rather similar to what we have observed for Figure 7.4, both in terms of pruning criteria and in terms of differences among the various ANN topologies employed. We can assess the effect of pruning on different VSR morphologies by comparing the rows of the figure, deducing that the biped and the worm are impacted in a substantially equal manner by pruning.

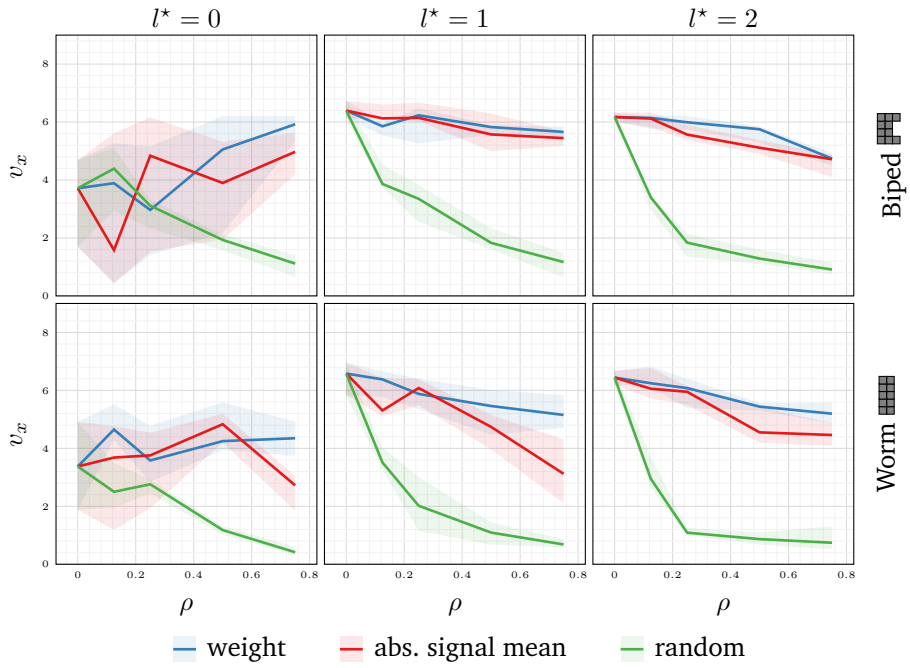


Figure 7.5: Fitness v_x (median and inter-quartile range across the 10 repetitions) vs. pruning rate ρ, for different pruning criteria (color), VSR morphologies (plot row), and ANN topologies (plot column).

As before, controllers pruned with a proper heuristic achieve results comparable to those who have not undergone pruning during their evolution. To confirm this, we perform a Mann-Whitney U test with the null hypothesis that, for each combination of VSR morphology, ANN topology, pruning criterion, and pruning rate ρ, the distribution of the best fitness is the same as obtained from the corresponding baseline controller, i.e., with the same VSR morphology and ANN topology, evolved without pruning, and we find that the p-value is greater than 0.05 in 30 out of 72 cases.

Pruning After the Evolution

Based on the results of Figure 7.4 and Figure 7.5, we speculate that controllers pruned with weight and absolute signal mean criteria are robust to pruning because they result from an evolution in which VSRs are subjected to pruning, rather than because those kinds of pruning are not particularly detrimental per se. To test this hypothesis, we carry out an additional experiment. We take the best individuals of the last generations for the centralized controller with the biped morphology and we re-assess them (on a randomly generated hilly terrain similar to the one used in evolution). For the individuals evolved without pruning, we perform $3 \cdot 4$ additional evaluations, introducing pruning after $t_p = 20\,\mathrm{s}$ with the previously mentioned 3 criteria and 4 rates ρ.

Figure 7.6 shows the outcome of this experiment, i.e., v_x on the re-assessment plotted against the re-assessment pruning rate ρ for both individuals evolved with (solid line) and without (dashed line) pruning. The foremost finding is that individuals evolved with pruning visibly outperform the ones whose ancestors have not experienced pruning, for almost all pruning rates. This corroborates the explanation we provided above, that is, VSRs whose ancestors evolved experiencing pruning are more robust to pruning than VSRs that evolved without pruning.

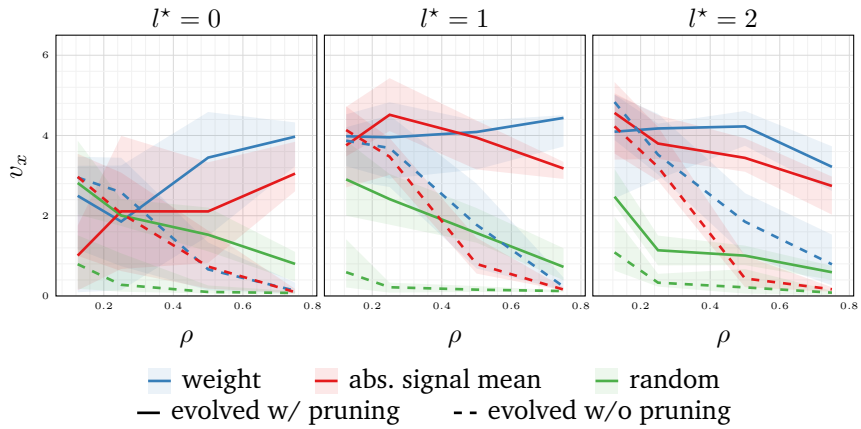


Figure 7.6: Median and inter-quartile range of re-assessment velocity v_x vs. re-assessment pruning rate ρ of individuals evolved with and without pruning for different ANN topologies for the centralized controller and biped morphology.

Besides analyzing the aggregate results, we also examine the behavior of a few evolved VSRs in a comparative way, i.e., with and without pruning in re-assessment. We find that, interestingly, in some cases the VSR starts to move effectively only after pruning: this might suggest that pruning shaped the evolu-

tionary path at the point that the lack of pruning becomes detrimental, similarly to what happens in the brain of complex animals (as described in Section 7.2). We provide videos of a few VSRs exhibiting a change in their behavior after pruning[1]. We speculate that choosing $t_i = t_p$ might be a contributing cause to this neat behavioral shift, hence in Section 7.4.4 we investigate the effects of setting $t_i < t_p$.

7.4.3 Impact on the Adaptability

For the sake of this question, we define VSR controllers as *adaptable* if they are able to effectively accomplish locomotion on terrains that none of their ancestors ever experienced locomotion on. Hence, to assess the adaptability of evolved controllers, we measure the performance in locomotion of the best individuals of the last generations on a set of different terrains. We experiment with the following terrains, also shown in Figure 3.6: (1) flat, (2) hilly with 6 combinations of heights and distances between hills, (3) steppy with 6 combinations of steps heights and widths, (4) downhill with 2 different inclinations, and (5) uphill with 2 different inclinations. As a result, each individual is re-assessed on a total of 17 different terrains. Note that, in this experiment, controllers are not altered in between evolution and re-assessment, i.e., they are re-evaluated with the same pruning criterion, if any, and pruning rate ρ as experienced during evolution.

Figure 7.7 displays the outcome of this experiment. Namely, for each of the different controller architectures, VSR morphologies, ANN topologies, and pruning criteria, the re-assessment velocity v_x (averaged on the 17 terrains) is plotted against the pruning rate ρ. The results in Figure 7.7 are coherent with the findings of Section 7.4.2: comparing the subplots, we can conclude that neither the controller architecture nor the morphology of the VSR are relevant in determining the effect of pruning on adaptability, whereas the ANN topology is somewhat of impact. Specifically, for shallow networks, pruning seems to enhance adaptability for the centralized controller architecture, whereas it has a slightly detrimental effect in all other cases.

Anyway, for controllers evolved employing weight or absolute signal mean pruning criteria, the re-assessment results are comparable to those of controllers evolved without pruning. We perform a Mann-Whitney U test with the null hypothesis that, for each combination of controller architecture, VSR morphology, ANN topology, pruning criterion, and pruning rate ρ, the distribution of the average re-assessment velocities across all terrains is the same as the one obtained from the re-assessment of the corresponding baseline controller, i.e., with the same VSR morphology and ANN topology, evolved without pruning, and we find

[1] `https://youtu.be/-HCHDEb9azY`, `https://youtu.be/oOtJKri6vyw`, `https://youtu.be/uwrtNezTrx8`

that the p-value is greater than 0.05 in 79 out of 144 cases.

7.4.4 Life-long Effectiveness

In the previous experiments, the behavior of the VSR before the pruning played no role in determining the fitness of the robot, hence in driving the evolution. To determine whether evolution can find a path towards VSR controllers that are life-long effective, we repeat the experimental pipeline described in Section 7.4.2, setting $t_i = 5\,\mathrm{s}$ and $t_p = 20\,\mathrm{s}$ for the velocity v_x calculation—we still "discard" the first $5\,\mathrm{s}$ of each simulation to avoid considering transient behaviors. This way, the VSR fitness is computed by taking into account both phases of the life of the VSR, before and after the occurrence of pruning. Such procedure has stronger biological resemblance than discarding the pre-pruning life for fitness computation, as in nature the survival and mating likelihoods are estimated on the entire lifespan of an individual, and not only after full brain development.

Since in Sections 7.4.2 and 7.4.3 we have noticed no significant differences between the controller architectures, VSR morphologies, and ANN topologies, for this analysis we only experiment with the centralized controller architecture with an ANN with one hidden layer on the biped morphology. Again, we perform 10 independent evolutionary optimizations, each based on a different random seed.

Figure 7.8 displays the results for this experiment (right plot), paired with those obtained with the corresponding configuration from Section 7.4.2 (left plot). Namely, for each pruning criterion the median and the inter-quartile range of velocity at the end of evolution v_x are plotted against the pruning rate. Observing the plots we can answer affirmatively to our third question: evolution has indeed managed to find a successful path towards controllers that can perform effectively both before and after pruning. Comparing the two plots of Figure 7.8 there are no outstanding differences, so we can draw similar conclusions as in Section 7.4.2, in the sense that pruning remains not significantly detrimental, provided that it is applied with proper heuristics and not randomly.

To gain further insights, we consider the velocity of the evolved VSRs separately for the two phases of life (before and after pruning). We compute the pre-pruning velocity using $t_i = 5\,\mathrm{s}$ and $t_f = 20\,\mathrm{s}$ and the post-pruning velocity with $t_i = 20\,\mathrm{s}$ and $t_f = 60\,\mathrm{s}$. The results are shown in Figure 7.9, where, similarly as before, the velocity is plotted against the pruning rate for each pruning criterion applied. In the plots, the solid line indicates the pre-pruning velocities, while the dashed line represents the post-pruning velocities. Comparing the two lines for each criterion, we can note that for the weight and absolute signal mean criteria there is a small gap between the two lines, which indicates that most of the controllers abilities are retained after pruning. Contrarily, for the random criterion there is a significant performance decrease after the occurrence of pruning,

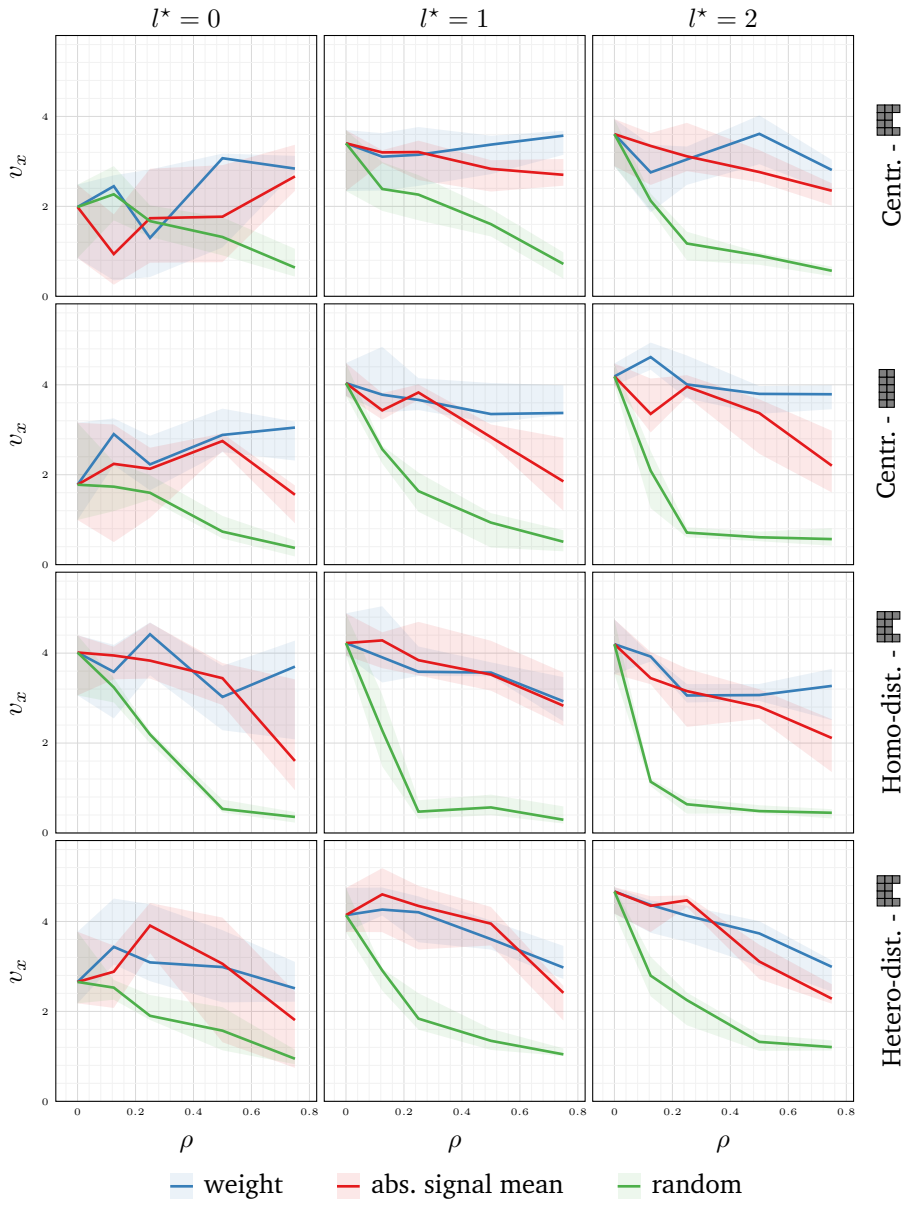


Figure 7.7: Median and inter-quartile range of re-assessment velocity v_x vs. pruning rate ρ averaged across re-assessment terrains for different pruning criteria, VSR controller architectures, VSR morphologies, and ANN topologies.

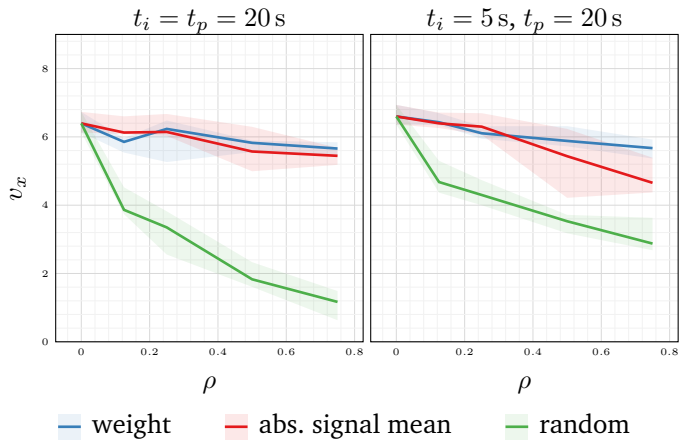


Figure 7.8: Fitness v_x (median and inter-quartile range across the 10 repetitions) vs. pruning rate ρ, for controllers evolved with $t_i = t_p = 20\,\mathrm{s}$ (first column) and controllers evolved with $t_i = 5\,\mathrm{s}$ and $t_p = 20\,\mathrm{s}$ (second column). Both controllers share the centralized architecture, the ANN topology with $l^\star = 1$, and the biped morphology.

which is in line with the previous findings (Section 7.4.2). We explain this result as follows: since random pruning acts differently on individuals of the same evolutionary lineage, evolution is not able to "guarantee" good performance after pruning. However, since it is driven also by the performance before pruning, evolution produces controllers that are effective in terms of pre-pruning velocity. Put simply, with random pruning only one of the two objective in a bi-objective evolutionary optimization is actually improvable.

Having observed that, similarly to biological organisms, controllers can evolve to be effective both before and after pruning, we decide to investigate the performance of such controllers also in terms of adaptability. To this extent, we repeat the experimental pipeline described in Section 7.4.3.

Figure 7.10 shows the re-assessment velocities for this experiment (right), paired with those obtained with the corresponding configuration in Section 7.4.3 (left). From the comparison of the two subplots, individuals that are evolved in a more biologically plausible fashion, i.e., taking into account both the pre- and post-pruning velocities for fitness evaluation, seem to be slightly more adaptable than those whose evolution was carried out considering only the post pruning performance. However, the difference between the two plots is not significant, hence we cannot draw sharp conclusions on this.

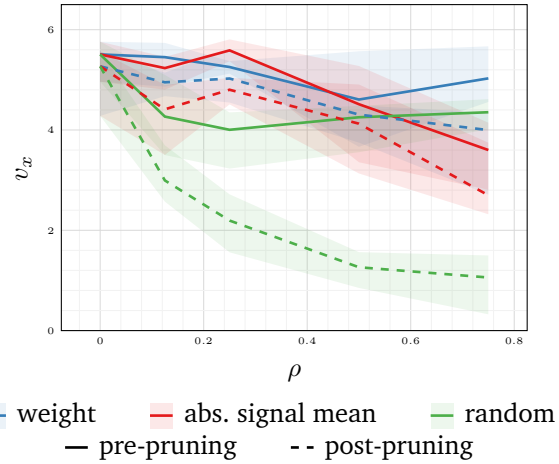


Figure 7.9: Median and inter-quartile range of velocity v_x vs. pruning rate ρ of individuals before (solid line) and after (dashed line) pruning for different pruning criteria (color).

7.4.5 Behavior Analysis

Having observed that pruning does not significantly affect the velocities of VSRs in locomotion, we decide to investigate its behavioral impact. Namely, we aim at systematically evaluating whether the pre- and post-pruning behaviors of VSRs are substantially different. To this extent, we rely on a consolidated data analysis procedure consisting of (1) feature extraction, (2) dimensionality reduction, and (3) visualization.

The features we employ to capture a VSR behavior in locomotion [263] are based on the movement of the center of mass of the VSR over time and on the way the VSR touches the ground during gait, i.e., its footprints. Here we provide just a brief description of the feature extraction procedure—we refer the reader to [263] for further details.

Concerning the center of mass movement, we extract its signals on the x- and y-axis from a sequence of snapshots—a snapshot is the description of the spatial configuration of every voxel of the VSR at a given time step—and we compute the signals of their first differences. From these, we calculate the FFTs, from which we take the magnitude and we filter out frequency components not in the range $[0\,\text{Hz}, 10\,\text{Hz}]$. Last, we re-sample the obtained signals to have $n_{\text{freq}} = 100$ components for each axis, constituting the final feature vector related to the center of mass movement. This part of the pipeline is similar to that employed in Chapter 3.

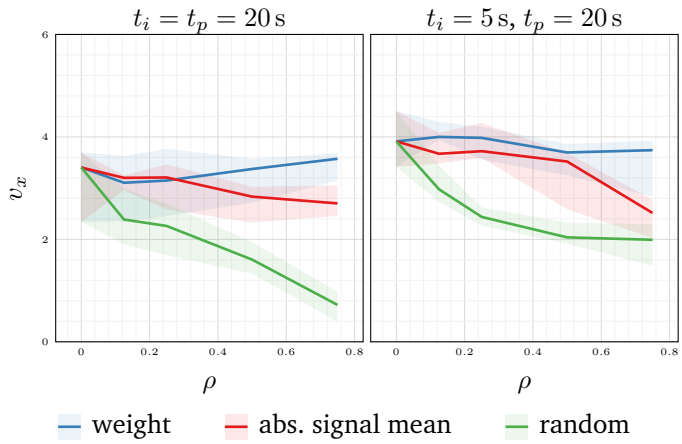


Figure 7.10: Average re-assessment velocity v_x (median and inter-quartile range across the 10 repetitions) vs. pruning rate ρ, for controllers evolved with $t_i = t_p = 20\,\mathrm{s}$ (first column) and controllers evolved with $t_i = 5\,\mathrm{s}$ and $t_p = 20\,\mathrm{s}$ (second column). Both controllers share the centralized architecture, the ANN topology with $l^\star = 1$, and the biped morphology.

7

Regarding the footprints, for each snapshot at each time step k, we project the minimal bounding square of the VSR on the x-axis, i.e., the smallest square parallel to the x-axis that completely contains the VSR, and we partition the projection in 8 equal sections, from which we build a binary vector (of size 8), the *footprint*, where each element is set to 1 if the VSR is touching the ground for more than half of the corresponding segment. To extract some features from the footprints, we consider a sequence of snapshots, which we process in the following way:

1. We split it in a sequence of non-overlapping subsequences, each corresponding to $\Delta t_{\text{footprint}} = 0.5\,\mathrm{s}$.
2. We compute the sequence M of footprints, where each element is obtained as the element-wise mode of the footprints in the corresponding subsequence;
3. We consider all the non-overlapping n-grams of footprints in M, $2 \leq n \leq 10$ occurring at least twice, we compute their overall duration and we select the main n-gram $M^\star$ as the one with the longest overall duration.
4. We process $M^\star$ to obtain the following descriptors: average touch area of the footprints in $M^\star$, overall duration of all $M^\star$ occurrences, length of the

main n-gram $|M^\star|$, mode of the intervals between consecutive occurrences of $M^\star$, and rate of intervals that are equal to the mode.

We obtain the feature vector related to the footprints of the VSR through the concatenation of the features extracted from $M^\star$.

Given the concatenation of the feature vectors of the center of mass movement and of the footprints, we perform dimensionality reduction using the PCA from 25 to 2 components, in order to visualize the results in scatter plots.

We exploit the aforementioned analysis pipeline to evaluate the impact of pruning on the behavior of a VSR. In addition, we investigate whether changing the beginning evaluation time t_i with respect to the pruning time t_p results in more or less visible behavioral differences. To this extent, we focus on the VSR configuration employed both in Section 7.4.2 and Section 7.4.4, namely, the biped morphology with the centralized controller architecture with an ANN with 1 hidden layer.

For each evolved VSR, we extract the behavior features in a re-assessment performed on flat terrain, in order to minimize the impact of any terrain irregularities on the features. To distinguish pre- and post-pruning behaviors, we perform the feature extraction on two separate intervals of the VSR lifetime: before pruning, from $5\,\mathrm{s}$ to $20\,\mathrm{s}$, and after pruning, from $20\,\mathrm{s}$ to $60\,\mathrm{s}$.

The results are shown in the scatter plots of Figures 7.11 and 7.12, for VSRs evolved with $t_i = t_p = 20\,\mathrm{s}$ (i.e., with the evolution driven only by post-pruning performance) and $t_i = 5\,\mathrm{s}$ and $t_p = 20\,\mathrm{s}$ (i.e., with the evolution driven by lifelong performance), respectively. Each point in the scatter plot corresponds to a behavior defined by the aforementioned features, its size depends on the velocity achieved by the VSR in the evaluation interval.

When comparing the two figures, the most notable observation is that in Figure 7.11, the distributions of behaviors post-pruning appear to diverge more from the pre-pruning behavior distributions. This is in contrast to Figure 7.12, where the two distributions generally exhibit greater overlap, especially at lower pruning rates. To explain this result, we consider that the VSRs in Figure 7.11 are evolved with $t_i = t_p = 20\,\mathrm{s}$, meaning that the fitness evaluation does not account for the robot behavior before pruning. We hypothesize that, under these conditions, evolution favors a post-pruning behavior that is reasonable but does not prioritize pre-pruning performance. Consequently, there is a more pronounced shift between pre- and post-pruning behaviors. In contrast, the VSRs in Figure 7.12 were evolved with $t_i = 5\,\mathrm{s}$, emphasizing consistent performance throughout the robot lifespan. This likely explains the smaller shift in behaviors, as the VSRs are selected to perform well both before and after pruning.

Focusing on Figure 7.12, it is evident that at higher pruning rates, the pre- and post-pruning distributions tend to diverge more noticeably. However, even for VSRs whose controllers are not subjected to pruning (first column, $\rho = 0$), there

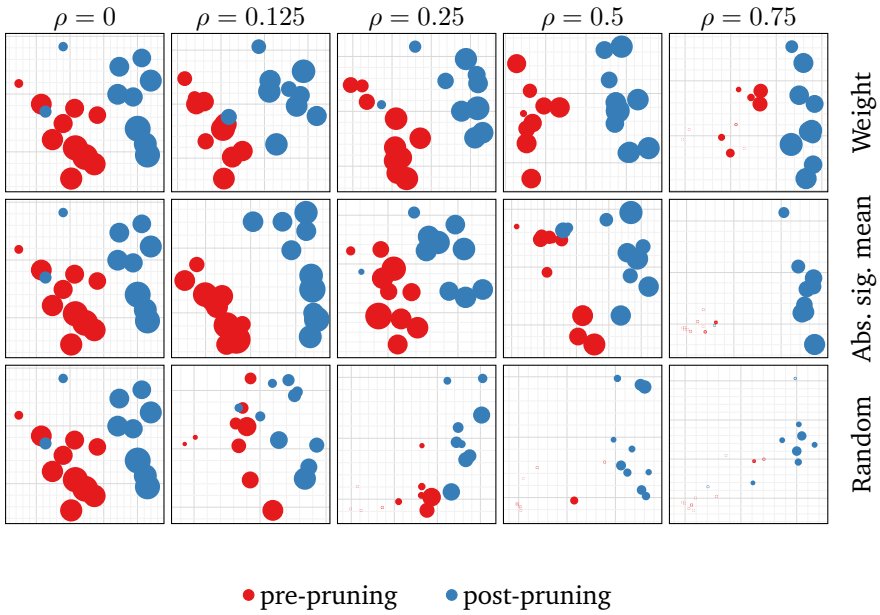


Figure 7.11: Scatter plots of the first two components resulting from the PCA analysis of the features described in Section 7.4.5 for the biped VSR with a centralized controller. Each subplot corresponds to a pruning criterion (row) and a pruning rate ρ (column); pruning is applied at $t_p = t_i = 20\,\mathrm{s}$. The color of the bubble indicates whether the evaluation of velocity and feature extraction are performed before or after the occurrence of pruning, while the size of the bubble is proportional to the achieved velocity v_x.

is still a small but notable shift in behavior. Additionally, the diversity among behaviors of different robots evolved and reassessed under the same conditions (i.e., markers of the same color within the same plot) is quite significant. Given these observations, it is difficult to identify a clear overall trend regarding the impact of pruning on the behavior of evolved VSRs.

Regarding the behavioral shifts caused by different pruning criteria, the plots do not allow us to draw any definitive conclusions. When comparing the rows of Figure 7.12, we observe that with pruning rates $\rho \leq 0.25$, the greatest deviation is induced by the absolute signal mean criterion, although the VSRs largely retain their functionality. However, at higher pruning rates, the shifts caused by all criteria appear to be similar. Despite this, VSRs pruned using a well-chosen heuristic tend to maintain their velocities, whereas those subjected to random pruning experience a notable decline in performance.

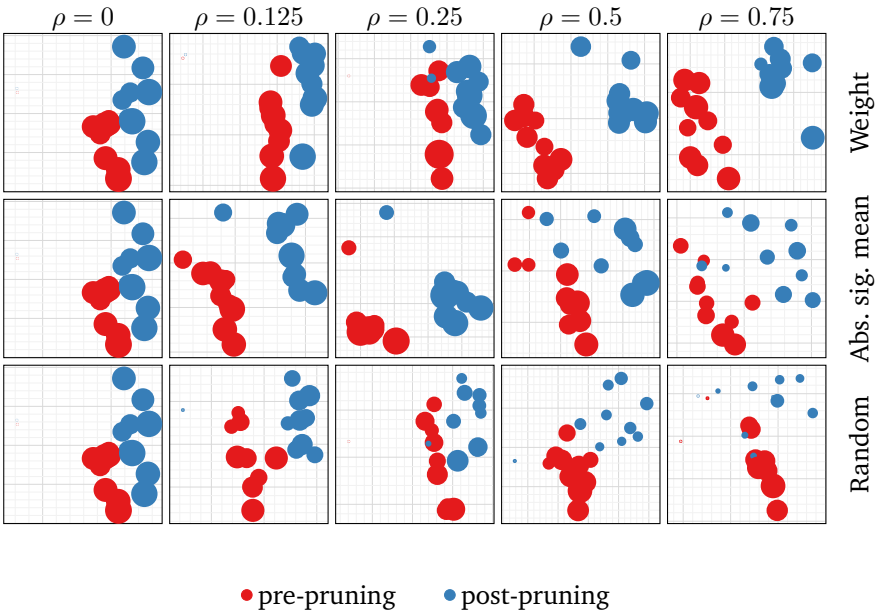


Figure 7.12: Scatter plots of the first two components resulting from the PCA analysis of the features described in Section 7.4.5 for the biped VSR with a centralized controller. Each subplot corresponds to a pruning criterion (row) and a pruning rate ρ (column); pruning is applied at $t_p = 20\,\mathrm{s}$ ($t_i = 5\,\mathrm{s}$). The color of the bubble indicates whether the evaluation of velocity and feature extraction were performed before or after the occurrence of pruning, while the size of the bubble is proportional to the achieved velocity v_x.

7.5 Concluding Remarks

In this chapter, we explored the impact of integrating pruning into the evolution of neural controllers for VSRs. Our goal was to determine whether this biologically inspired technique could enhance adaptability in artificial agents, similar to its effects in living organisms, or if it would be detrimental to the evolved individuals. To achieve this, we focused on the task of locomotion and evolved the controllers for VSRs using various pruning criteria and rates. Our study encompassed three types of controller architectures, two VSR morphologies, three ANN topologies, and two types of fitness measures (post-pruning and life-long). We also performed a behavioral analysis, examining frequency-domain and gait features.

Our results showed that applying pruning at a moderate rate with an appropriate criterion during evolution can produce individuals comparable to those evolved without pruning, and can even enhance their robustness to pruning. Additionally, controllers evolved with pruning demonstrated similar adaptability to new tasks, such as locomotion on unfamiliar terrains, compared to those evolved without pruning. Thus, we concluded that pruning can effectively reduce network complexity without compromising the performance of the evolved controllers.

II

Exploring the Notion of Interpretability in the Control of Embodied Agents

In this part of the thesis, we delve into the concept of interpretability, specifically examining how to achieve interpretability in the control of embodied agents without compromising their performance.

We begin by focusing on the subjective nature of interpretability in Chapter 8. Using a human-in-the-loop system with both real and simulated users, we investigate the factors that influence the perceived interpretability of a model. Then, in Chapters 9 and 10, we utilize graph-optimization techniques to synthesize interpretable controllers for various types of robots. In the former chapter, we compare a bi-objective optimization approach that balances performance and interpretability against a performance-focused optimization where interpretability is expected to naturally emerge. In the latter chapter, we explore solution diversity as a means to simultaneously enhance performance and interpretability.

8

Interpretable Symbolic Regression Models with Human-in-the-Loop

Interpretability is essential for ensuring the fair and responsible application of Machine Learning (ML) in high-stakes scenarios. Genetic Programming (GP) is often employed to create interpretable ML models due to its operation at the level of functional building blocks. However, the interpretability of a model can vary depending on the observer. To address this, we explore a human-in-the-loop system that allows users to guide the GP generation process according to their preferences, which are learned online by an Artificial Neural Network (ANN). First, we develop more generalized representations of ML models for the ANN to learn from, broadening the system applicability to various problems. Second, we conduct a detailed analysis of the system components. Specifically, we propose an incremental experimental evaluation to: (1) assess how effectively an ANN can capture perceived interpretability from simulated users, (2) examine how different simulated user feedback profiles influence the GP outcomes, and (3) determine if human participants prefer models generated with their involvement over those without. Our findings provide insight into the advantages and limitations of using a human-in-the-loop approach to discover interpretable ML models with GP.

This chapter is based on the following publication: **G. Nadizar**⋆, L. Rovito⋆, A. De Lorenzo, E. Medvet, and M. Virgolin (⋆ shared first co-author). An analysis of the ingredients for learning interpretable symbolic regression models with human-in-the-loop and genetic programming. *ACM Transactions on Evolutionary Learning*, 2024 [303].

8.1 Introduction

In recent years, it has become evident that irresponsible use of ML models can pose significant risks [176]. To address this issue, the field of Explainable Artificial Intelligence (XAI) focuses on methods for explaining model predictions and enhancing interpretability, as well as developing models that are inherently interpretable [2, 134]. The need for XAI research is accentuated by the presence of highly complex opaque models in high-stakes real-world applications, where users must thoroughly interact with these models to analyze and understand their predictions. Current consensus suggests that methods for explaining opaque models have limitations, depending on their underlying assumptions, and should be used cautiously [282, 283]. If the data permits the discovery of an interpretable model that achieves high accuracy, it is generally preferable to use this interpretable model over an opaque alternative [367].

GP is an Evolutionary Computation (EC) algorithm used to find interpretable models, such as formulae, Decision Trees (DTs), rule sets, or computer programs [202]. GP operates by initializing a population of (typically) random models made up of basic operations or instructions, and then evolving this population through stochastic recombination or mutation, with survival of the fittest determining which models are retained. If the basic operations involved are clearly understandable and GP discovers an accurate model, there is a chance that the resulting model is also interpretable, meaning it consists of a limited number of operations arranged in an understandable way. However, relying solely on chance to discover interpretable models is unlikely to yield satisfactory results.

To guide GP towards finding interpretable models, we need to define an objective function that serves as a proxy for interpretability. Unfortunately, interpretability is inherently subjective—it varies based on the specific application and the user's background [31, 235]. Additionally, different applications might require balancing model accuracy with interpretability in various ways, depending on the context and stakes involved [146, 122].

Virgolin et al. [447] tackled the challenge of the subjective nature of interpretability using a GP-based human-in-the-loop system known as *Model Learning with Personalized Interpretability Estimation (ML-PIE)*. This system employs GP in a multi-objective setting, where one objective is the accuracy of the model and the other is a proxy for interpretability, represented through an ANN. The ANN is trained using feedback from the user, which is provided via a Graphical User Interface (GUI) during the GP evolutionary process. Specifically, users are shown pairs of models (from those generated by the ongoing GP process) and are asked to indicate which model they find more interpretable based on their personal understanding of interpretability. Over time, the ANN learns to approximate user preferences, and users tend to favor models generated by ML-PIE compared to

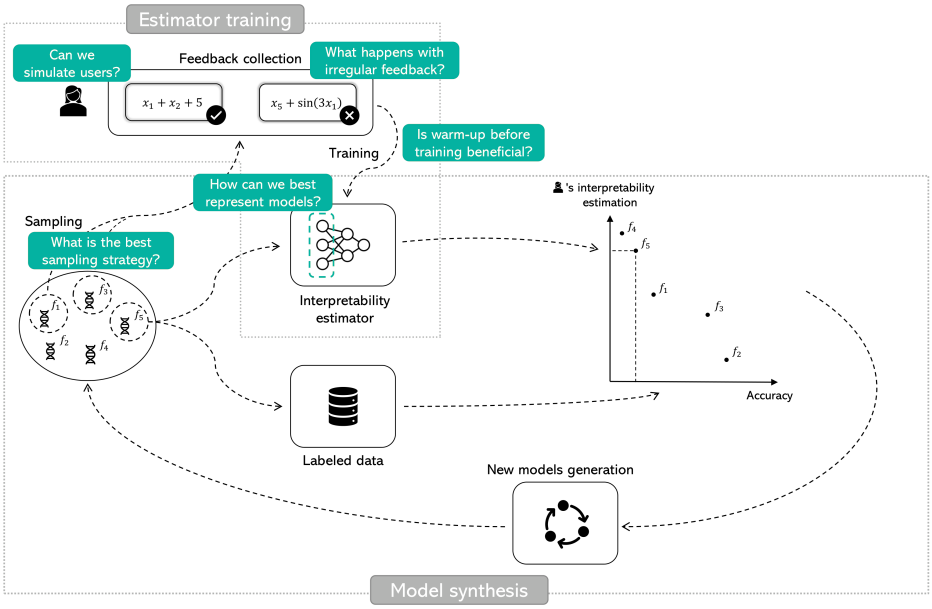


Figure 8.1: Schematic view of ML-PIE. In the green rectangles, we highlight the questions we tackled in this study.

those produced by non-personalized methods. Figure 8.1 illustrates ML-PIE, detailing its main components and phases.

While the proposal by Virgolin et al. [447] shows promise, it has several limitations. Firstly, the system, particularly the ANN component, was specifically designed for Symbolic Regression (SR) models. For instance, it uses features such as the count of basic arithmetic operations (e.g., $+$, $-$, $\times$, $\div$). It is uncertain whether ML-PIE would perform effectively with more general features applicable to other types of models, such as DTs or rule sets. Secondly, Virgolin et al. [447] did not address several key issues. For example, it is unclear whether the reliability of the proposed Active Learning (AL) strategy [356, 383] scales with the complexity of the interpretability representation used, particularly as it becomes more detailed. Additionally, it is unknown whether updating the ANN as the GP search progresses has a positive or negative effect on the search outcome, potentially influenced by the frequency of user feedback.

In other words, there is still much to learn about how best to integrate human input into GP for interpretable ML and what limitations this integration might have. Thus, this chapter aims to advance our understanding of using GP with a human-in-the-loop approach for interpretable ML.

Specifically, we address the aforementioned shortcomings by: (1) proposing alternative representations (features) to capture the interpretability of models evolved by GP, and (2) experimentally evaluating the components of ML-PIE, both with real and simulated users. Figure 8.1 provides a detailed view of the specific components of ML-PIE that we investigate and outlines the aspects we target in the form of questions.

We organize our experimental evaluation into three phases, as follows.

1. First, we examine the expressiveness of different model representations and how effectively an ANN can learn from them based on simulated user feedback. We identify a trade-off between expressiveness and learning capability and demonstrate that uncertainty-based AL (as proposed in [447]) does not scale well with larger representations unless there is a substantial increase in the volume of feedback.

2. Next, we integrate the ANN into GP as a proxy for user interpretability and analyze the search dynamics under various scenarios. Simulated users provide feedback based on ground-truths. We find that different notions of interpretability affect the search outcomes, but the system remains robust against inconsistent feedback from users who provide input irregularly.

3. Finally, we engage actual human participants to use our system for identifying suitable models for two real-world datasets. Participants assess, in a blind manner, whether the models generated with their involvement are preferable to predefined models. Our findings indicate that with limited user feedback, participants often prefer the predefined models over those generated by our system, suggesting that a longer feedback collection period might be necessary.

8.2 Related Works

Recent advancements in explaining and interpreting ML model predictions have garnered significant attention in scientific literature [159, 95, 130, 157, 140]. The most relevant strategies that can be adopted to make models explainable consist in directly synthesizing interpretable models by means of *ad hoc* learning algorithms or, alternatively, applying specifically designed techniques (e.g., reverse engineering, feature attribution methods, counterfactual explanations) on opaque models to obtain the needed explanations [359, 134, 2, 440, 462]. The latter approach can be model-agnostic, but has the clear limitation that it cannot provide a complete understanding of how a model operates for any potential input [367, 235]. Here, we concentrate on the first approach, focusing on the synthesis of models that are inherently interpretable.

In general, DTs, rule-based systems, and linear models are often considered better suited for generating interpretable models compared to more complex models like deep neural networks [369, 97, 135, 166, 373, 79]. However, even these seemingly simpler models require justification, as simplicity alone does not guarantee interpretability [235, 199]. For DT models, simplification can be achieved through techniques such as size reduction and pruning [171, 46] (as seen also for the robotics case in Chapter 7), or by optimizing a loss function that balances detection accuracy with model complexity [212, 406]. Linear models can be simplified by reducing the number of features [482, 429, 419, 341]. Overall, many types of ML models can benefit from removing irrelevant and redundant features, including linear models, regression trees, and DTs. Additionally, ANN-based models can be improved by eliminating redundant hidden layers and neurons.

In the realm of interpretable ML, Evolutionary Algorithms (EAs) like Genetic Algorithms (GAs) and GP have garnered significant interest for their ability to generate potentially interpretable models (EC for XAI) [18, 114, 384, 266]. GP, in particular, is frequently used to create inherently interpretable models such as DTs [108] and classification rules through Learning Classifier System (LCS) algorithms [428]. A common strategy for achieving interpretable trees involves constraining the complexity of the tree. This can be accomplished by minimizing the number of nodes while pursuing other objectives and by restricting the function set to simpler functions only [364, 55, 101, 391, 222, 221, 295, 444, 47].

Model components can also be assigned weights according to a predefined weighting scheme, allowing the weighted sum of these components to estimate the interpretability of the model: models with lower interpretability are penalized during the evolutionary process [150, 259]. Additionally, formula simplification techniques can be applied, potentially as a post-processing step, to further refine the learned models, improving their interpretability and readability [173].

Recent research has made significant strides in the field of interpretable EC, applied to Reinforcement Learning (RL) scenarios [181, 458, 439]. Custode and Iacca [71] combined Grammatical Evolution (GE) [322] with Q-learning [454] to evolve interpretable DTs for RL problems, focusing on learning a decomposition of the state space. This approach was later extended by the authors to accommodate continuous action spaces [72]. Given that interpretable models often struggle with raw data and high dimensionality, Custode and Iacca [73] introduced a framework that employs end-to-end pipelines consisting of multiple interpretable models, optimized using EC. This system allows for the decomposition of the decision-making process in an RL environment and the extraction of high-level features from raw data, which are easier to understand. Additionally, ML techniques have also been utilized to develop policies for managing pandemics [195, 424, 273, 74].

Another proposed strategy involves a data-driven approach that derives an interpretability formula from human feedback, which can then be used to estimate the interpretability of formulae synthesized during a GP evolutionary process [445]. This method was subsequently utilized by [71]. The key difference between [445] and our approach is that while [445] learns an interpretability formula by training a linear model on data collected through a mathematically constrained survey, our framework aims to learn an arbitrary notion of interpretability tailored to the preferences of any user, regardless of their domain of expertise.

The framework introduced by Virgolin et al. [447] addresses the subjectivity of interpretability by implementing a SR system where formulae are assessed based on an interpretability estimator trained with user feedback. This system employs a bi-objective GP process, where one objective evaluates model accuracy, while the other assesses interpretability through an ANN trained alongside the evolutionary process. In this system, pairs of formulae are selected using an AL criterion based on the estimator uncertainty and presented to users [466, 383]. Users then choose the more interpretable formula from each pair based on their subjective concept of interpretability. This feedback trains the interpretability estimator, enabling it to approximate a personalized Proxy of Human Interpretability (PHI).

Several other studies have explored incorporating human feedback during the training of Artificial Intelligence (AI) systems with a human-in-the-loop approach [251, 382, 64]. Murphy et al. [295], suggesting that integrating human feedback during the model synthesis process can aid in discovering interpretable models. Additionally, there are works where AL is used within GP [23, 83, 169] to directly incorporate collected labels into the objective function.

8.3 Personalized Model Learning with ML-PIE

We consider supervised learning where, given a dataset $D = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_i$ of observations and corresponding labels, the goal is to find a model $f : X \to Y$ that is *accurate*, i.e., $f\left(\boldsymbol{x}^{(i)}\right)$ correctly predicts the label $y^{(i)}$ for each i, and is also *interpretable*.

We assume that the accuracy of a model can be measured with a function $q_f : \mathcal{F}_{X \to Y} \to \mathbb{R}$, where $\mathcal{F} = \{f, f : X \to Y\}$ is the *space of models* (or hypothesis space). In practice, a number of well-defined accuracy measures exists, such as the mean squared error or the coefficient of determination for regression (i.e., when $Y = \mathbb{R}$), and the percentage of correct classifications for classification (i.e., when Y is a finite set without intrinsic ordering). Regarding interpretability, we assume that a measure $\hat{\psi}_f : \mathcal{F}_{X \to Y} \to \mathbb{R}$ exists that captures the *personal* notion

of interpretability of the user.

We consider the case in which the user wants to solve a supervised learning problem by providing the dataset D and the desired measure of accuracy q_f, but not their interpretability measure $\hat{\psi}_f$ (e.g., because the user is *unable* to formalize $\hat{\psi}_f$). We assume, however, that the user is available for providing feedback (annotations) about the interpretability of models that are generated by the ML process.

8.3.1 Framework Overview

We propose to solve the interpretable model learning problem as a bi-objective search in the space $\mathcal{F}_{X \to Y}$ of models, where accuracy and interpretability are the two objectives. Since the *true* interpretability measure $\hat{\psi}_f$ is not provided, the corresponding objective is pursued using an *estimator*[1] ψ_f of interpretability that is learned during the search of the model using the user's feedback. This approach is an extension of the one pursued by Virgolin et al. [447] and, just like in the original work, we refer to it as *ML-PIE*.

Internally, ML-PIE resorts to a *model search algorithm* for searching $\mathcal{F}_{X \to Y}$ and to an *AL algorithm* for training the estimator. Both algorithms are iterative in nature. The training process of the estimator is implemented via *human-in-the-loop*: the user is prompted with selected models among the ones being evaluated by the search algorithm, and asked to provide feedback about the models' interpretability. ML-PIE collects the user's feedback concurrently with the model search process, yet asynchronously: that is, the rate at which models are generated is independent from the rate at which the user provides feedback on (part of) them.

8.3.2 Concurrent Model Search and Active Learning

In ML-PIE, the model search and AL algorithms operate concurrently (though asynchronously). This concurrent approach is preferred over an offline approach, where user feedback is collected *before* the model search begins. The advantage of the concurrent method is that users are presented with models that are actively being discovered by the model search algorithm. Consequently, ψ_f is trained on data that is *in-distribution* w.r.t. the model search, as this data reflects the actual distribution of models being generated. In contrast, an offline approach would lack a guarantee that the user feedback covers the same distribution of models as those being discovered by the model search algorithm.

[1] Although both the model f that evolution is searching for and the estimator of interpretability ψ_f can be called model or estimator, from here on we refer with *model* to the first, and with *estimator* to the second.

At initialization, ML-PIE starts with a randomly generated initial estimator ψ_f. Note that ψ_f can also be initially set to be an estimator that was *pre-trained*, e.g., on previous data that was annotated by one or multiple users [445], or on a proxy of interpretability from the literature, e.g., [448]. We discuss a few alternatives for initializing ψ_f in Section 8.4.3. Then, at each iteration, the model search algorithm builds some new candidate models and evaluates them using q_f and the current ψ_f. ML-PIE adds (syntactically unique) models discovered by the model search algorithm to an initially empty set $F \subset \mathcal{F}_{X \to Y}$ of models. F is re-set at every iteration of the model search algorithm, to contain only models that are relevant to the current status of the search. Meanwhile, the AL algorithm is in charge of iteratively refining ψ_f. At each iteration, the AL algorithm builds a *query*, i.e., it selects a pair f_1, f_2 of candidate models from F according to a priority rule (see Section 8.4.4), and submits the two models for user assessment. The user is shown the two models on a GUI and is instructed to choose the one that they believe to be more interpretable. When the user answers the query, the AL algorithm updates ψ_f by using the feedback as signal. Once ψ_f has been updated, the AL algorithm builds a new query, thus starting a new iteration of the feedback collection phase.

We note that feedback collection in ML-PIE is simplified to a binary choice between two models, making the annotation task more manageable for users and applicable to various model spaces $\mathcal{F}_{X \to Y}$. Asking users to rank more than two models simultaneously would likely be too demanding. Likewise, requiring users to rate a single model on a numerical scale, such as from one to ten, could also be challenging for them. Additionally, creating a suitable scoring function for users to evaluate models can be a complex task for the system designer.

The feedback collection process (or AL algorithm) runs concurrently with the model search algorithm until the latter concludes. The model search algorithm terminates when it reaches a predefined budget, which could be a maximum number of iterations (such as generations in GP) or a specified time limit.

In summary, ML-PIE runs two iterative algorithms concurrently. The model search algorithm updates one or more models at each iteration, guided by a fixed q_f and a dynamically evolving ψ_f; the pace of iterations is the fastest possible on the machine ML-PIE is executing on. Meanwhile, the AL algorithm updates ψ_f with user feedback at each iteration, with the iteration pace dependent on how quickly the user provides feedback.

8.3.3 Applicability

ML-PIE, as described so far, is a rather general approach that can be applied to many different use cases. For example, regarding the space of models $\mathcal{F}_{X \to Y}$, the only requirements are that (1) models in $\mathcal{F}_{X \to Y}$ are compatible with the

estimator being updated by the AL algorithm ψ_f, and that (2) pairs of models can be judged by the user in terms of relative interpretability. In practice, the first requirement can be met if a proper encoding of the models in $\mathcal{F}_{X \to Y}$ is used—we discuss a concrete case in Section 8.4. The second requirement can be met if the models can be visualized: hence, the user's feedback can be collected by means of a GUI.

For what concerns the model search algorithm, i.e., the one searching in $\mathcal{F}_{X \to Y}$, we require it to be bi-objective, although this constraint can, if necessary, be relaxed using linearization or lexicographic order among objectives.

Lastly, a practical requirement is that the speed at which the model search algorithm progresses must be approximately compatible with the speed at which the user provides feedback. In other words, the model search algorithm should run for enough time for the user to provide a reasonable amount of feedback (we experiment in detail on this in Section 8.5.4).

8.4 ML-PIE for Symbolic Regression

We now consider a concrete application of ML-PIE to the case study of SR, showing how the general framework described so far can be adapted to this specific problem of interpretable ML.

SR is a form of supervised learning in which, given data on d independent numerical variables $(x_1, \ldots, x_d) = \boldsymbol{x}$ and on the dependent numerical variable y, one wishes to find the model $f : \mathbb{R}^d \to \mathbb{R}$ that best explains y from $\boldsymbol{x}$ while, *crucially*, f can be written as an *analytical expression* (or, simply, a formula). In other words, f must be realized by composition of basic and interpretable functional building blocks such as $+, \times, -, \div, \exp, \ln, x_1, x_2, 0.5, -1, \pi$, and so on. These building blocks are decided by the user of the system and are an input for the SR problem, which is NP-hard [442]. With respect to the general formulation of Section 8.3, here $X = \mathbb{R}^d$, $Y = \mathbb{R}$, and the space of models $\mathcal{F}_{X \to Y}$ is the space $\mathcal{S}_d$ of formulae for d variables. Consequently, the interpretability estimator $\psi_f : \mathcal{S}_d \to \mathbb{R}$ takes a formula as input.

We consider GP to act as model search algorithm since it has been shown to be an effective approach for SR [211]. We set GP to work in a bi-objective manner, with accuracy and interpretability as objectives. At termination, GP outputs a collection of formulae, with different trade-off levels between the considered objectives. This allows the user to choose the model that best fits their needs. We provide a more detailed description of the model search via GP in Section 8.4.1.

Concerning the AL algorithm for the interpretability estimator ψ_f, we investigate several design options. All of them share the following general structure. We use an encoding $h : \mathcal{S}_d \to \mathbb{R}^m$ to map a model, i.e., a formula, to a numeric

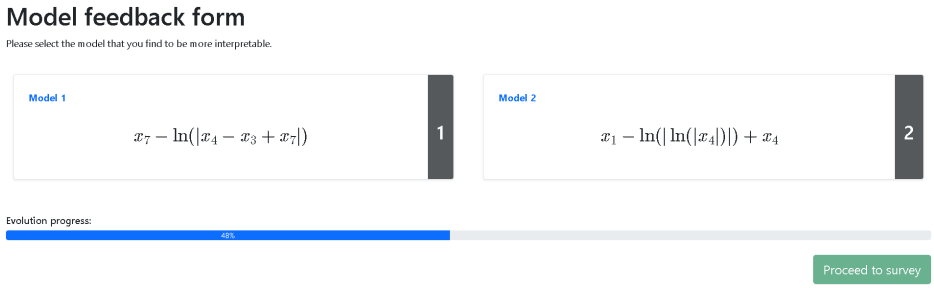


Figure 8.2: A screenshot of the GUI during the feedback collection process.

vector whose m components are features of the formula. These features must be decided beforehand, we discuss this in Section 8.4.2. Next, we use an ANN to obtain an interpretability estimate $\psi_f(f)$ for a formula f by giving the feature vector of f as the input for the ANN, i.e., $\psi_f(f) := \text{ANN}_{\boldsymbol{\theta}}(h(f))$, where $\boldsymbol{\theta} \in \mathbb{R}^p$ represents the parameters of the ANN. Over time, by using an AL approach, the parameters $\boldsymbol{\theta}$ must be optimized to make the ANN consistent with the user's feedback; provided that the choice of the encoding function h is adequate, i.e., the features of the formulae are informative. We discuss (and experiment with) a few design options for the encoding in Section 8.4.2 and for the ANN optimization in Section 8.4.3. Besides updating ψ_f by optimizing $\boldsymbol{\theta}$, the AL algorithm is also in charge of building the queries, i.e., of selecting which pairs of formulae should be shown to the user. We discuss design options concerning query building in Section 8.4.4. From a practical point of view, we collect the user's feedback via a GUI, depicted in Figure 8.2: ML-PIE shows two rendered formulae and the user can tell which one is the most interpretable with a single click on the GUI.

We emphasize that, while our experiments focus specifically on SR, our approach is broadly applicable as long as the few requirements outlined in Section 8.3.3 are met. For instance, the same methodology could be applied to RL problems where symbolic policies are sought, by incorporating a suitable fitness evaluation in simulated environments. Additionally, our approach can be generalized to the search for other types of models, not limited to symbolic formulae, as detailed in Section 8.4.2.

8.4.1 Bi-Objective Model Search

We use Non-dominated Sorting Genetic Algorithm II (NSGA-II) [86] (see Algorithm 2.3) as bi-objective model search algorithm, with the implementation provided in [35]. Specifically, we resort to a GP version of NSGA-II, encoding the formulae as trees [326]. We remark that the specific optimization algorithm is

not the key point of this work, rather the important part is how we compute the interpretability estimation. As a matter of fact, this system may be implemented as well by replacing NSGA-II with other multi-objective optimization algorithms. We believe that NSGA-II is the most convenient choice considering the scope of this work because of its popularity and proven speed and effectiveness for multi-objective discrete optimization tasks.

We leverage Algorithm 2.3 with the following specifications. For the initialization of the population, i.e., for the `init()` function, we use the *ramped half-and-half* initialization method [246], with a maximum depth of $d_{\text{max}} = 4$ (see Section 2.3.2). For the variation operator, i.e., for the `variate()` function, we combine subtree crossover and subtree mutation. First, we apply crossover to generate $90\,\%$ of the offspring, while we simply clone the sampled parents for the remaining $10\,\%$. Then, $60\,\%$ of the offspring, chosen at random, undergo subtree mutation. During this process, we discard (and repeat the creation of) any offspring with depth greater than $d_{\text{max}} = 4$ (as too large formulae are hardly interpretable), and that are identical to an existing member of the population or of the offspring generated so far. We discard identical trees because diversity preservation can greatly improve the performance of NSGA-II when used for GP [236]. Specifically, we retain two trees to be identical if they are semantically equivalent (e.g., $x_1 + x_2$ is the same as $x_2 + x_1$). We do this by simply comparing the corresponding predictions on the considered data, meaning that if the predictions are the same, then we assume that the trees are identical, even if they are syntactically different.

We employ the following primitives as nodes for the trees representing the formulae: (1) the variables $x_1, \ldots, x_d$ of the problem at hand, also known as the *features* of the dataset, (2) random constants sampled uniformly at random between -5 and 5, and (3) the mathematical operators $+$, $-$, $\times$, $\div^*$, $(\cdot)^3$, $\ln^*$, $\max$ (where * denotes the protected version[2] to avoid mathematical operations performed on out-of-domain values). We set $n_{\text{pop}} = 200$, $n_{\text{tour}} = 200$, and $n_{\text{evals}} = 10\,000$. These settings are the result of several preliminary experiments, and correspond to a reasonable compromise between performance and execution time. In particular, we aim to enable the algorithm to discover accurate models, while keeping the model search within reasonable time for the involved user. With the chosen settings, each model synthesis lasts approximately $10\,\text{min}$.

As already mentioned, NSGA-II aims to optimize accuracy and interpretability simultaneously. Regarding the former, we use the coefficient of determination, i.e., the R^2 score, to be maximized—which corresponds to maximizing the model accuracy. We include *linear scaling* [185, 186] to adapt the fitting of the synthesized formulae, given its proven effectiveness on real-world datasets [443]. Regarding interpretability, we rely on the outcome of the estimator ψ_f, which

[2] $\div^*(a, b) = \text{sign}(b)\ \frac{a}{|b| \text{ if } |b| > 10^{-9} \text{ else } 10^{-9}}$; $\ln^*(a) = \ln(|a| + 10^{-9})$

should ideally capture the perceived interpretability of a formula for the user in question. Since the estimations of interpretability change over time as ψ_f is updated by the user's feedback, the interpretability of the formulae is re-computed at every generation. Finally, the outcome NSGA-II is a set $S^\star \subset \mathcal{S}_d$ of formulae that achieve a Pareto trade-off between the two objectives, i.e., each formula corresponds to some level of compromise between accuracy and interpretability.

8.4.2 Model Encoding

We consider three options to realize the encoding function $h : \mathcal{S}_d \to \mathbb{R}^m$, which differ in terms of size, complexity, and expressiveness. In our setting of SR, as formulae are represented with trees, h operates on trees. We remark that the tree representation can be generalized to other problems besides SR, as, e.g., regular expressions [22] or DTs, which proves the generality of our approach.

Counts Encoding

With this option, a tree is encoded as a vector where each component corresponds to a possible primitive, and the value of that component is the number of times the primitive occurs in the tree. In particular, $|H|$ components are devoted to counting the occurrences of each mathematical operator, with $H = \{+, -, \times, \div^*, (\cdot)^3, \ln^*, \max\}$ for SR, d components are employed for counting the occurrences of each of the features of the problem, and one component counts how many numerical constants appear in the tree. The encoding vector further includes three components that provide additional information about shape and size of the tree, namely: (1) the ratio between the number of depth levels and the number of nodes; (2) the ratio between the maximal number of operands that an operator in the given tree can have, and the maximal breadth (i.e., the maximum number of nodes at a same depth level); (3) the percentage of leaf nodes.

The size of this encoding is thus $m = |H|+d+1+3$. This encoding can be advantageous because it scales with H and d, i.e., it is independent of the maximal number of nodes in the tree. However, this encoding does not distinguish between primitives appearing at different positions in the tree, which might impact interpretability.

One-hot Encoding

Different from the counts encoding, the one-hot encoding takes into account the exact position of each element in the tree. More specifically, each node is one-hot encoded as a vector of size $|H|+d+1$, where all components are set to zero with the exception of the one corresponding to the element in the node (the last

$+1$ being the vector element reserved to all the constants), which is set to one. Then, $h(f)$ is the concatenation of the encoding of all possible nodes as the tree is traversed from the root in a breadth-first fashion. To ensure encoding coherence, we assume all nodes to have a number of operands equal to the maximal possible (maximum arity α): in case a node is missing, i.e., it is an intron, the corresponding encoding is $\mathbf{0} \in \mathbb{R}^{|H|+d+1}$.

Clearly, the one-hot encoding is more expressive than the counts encoding, as it can capture a wide variety of properties of the tree that can be related to interpretability. However, the one-hot encoding scales poorly compared to the counts encoding, since it depends on $|H|$, d, as well as on the maximum possible number of nodes contained in a tree of maximal depth d_{max}. Namely, the encoding size is $m = \frac{\alpha^{d_{\text{max}}}-1}{\alpha-1}(|H|+d+1)$.

Level-Wise Counts Encoding

This last encoding represents a compromise between the counts encoding and the one-hot encoding. This encoding counts the occurrences of operators, problem features, and constants separately for each depth level $l \leq d_{\text{max}}$ in the tree. In addition, as for the counts encoding, we provide three descriptive components for the shape and size of tree.

The size of the level-wise counts encoding is $m = d_{\text{max}}(|H|+d+1)+3$, which makes it still fairly compact w.r.t. the maximal number of levels d_{max}, the size of the function set $|H|$, and the amount of problem features d. However, differently from the one-hot encoding, information of specific node positions is now missing.

Figure 8.3 shows how the size m of the three encodings scales for different combinations of $|H|$, d, α, and d_{max}.

8.4.3 ANN Optimization

We use an ANN as interpretability estimator. The ANN takes in input the encoded formula and returns a score of interpretability, i.e., $\psi_f(f) = \text{ANN}_{\boldsymbol{\theta}}(h(f))$. We make use of dropout [398] at prediction time [285] to make ANNs provide a measure of prediction uncertainty, which enables to perform uncertainty-based AL. Other ways of obtaining an uncertainty measure from an ANN could be used, e.g., bootstrapping or conformal prediction [346]: we choose dropout because it integrates well with the rest of our framework.

Regarding the architecture of the ANN, we employ an Multi-Layer Perceptron (MLP) with two hidden layers, each with 100 neurons with Rectified Linear Unit (ReLU) activations [304]. The output layer consists of a single node with $\tanh$ activation. Moreover, we add dropout to each hidden layer [398] with a probability of 0.25. Dropout is enabled both at training and evaluation time. During

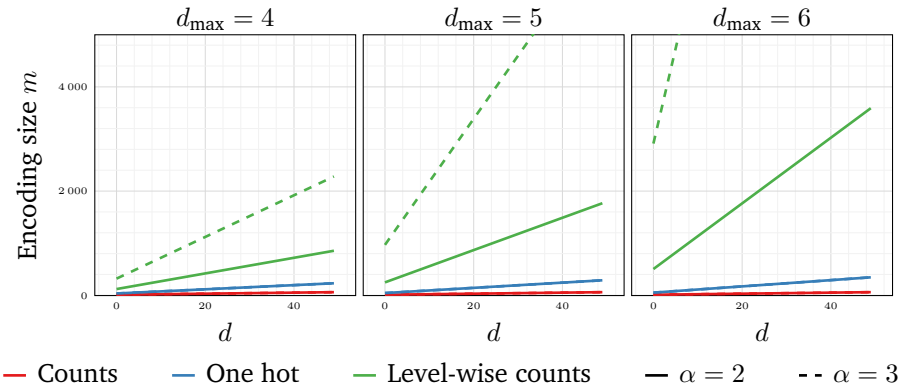


Figure 8.3: Encoding sizes (in different colors) w.r.t. the amount of problem features d (on the x-axis), the maximum tree depth $d_{\max}$ (in each subplot), and the maximum node arity α (solid vs. dashed line). We keep the size of the function set H fixed to 7, as in Section 8.4.1.

training, the ANN is applied (forward pass) to the encoding of the given formula *once* in order to obtain the signal needed to train the ANN parameters (explained below). Instead, when the ANN is used for inference, we perform $k = 10$ predictions (forward passes) for the same formula, to account for the stochastic nature of dropout. We then take the mean of the k predictions as interpretability estimate, and the standard deviation as uncertainty (see Section 8.4.4).

We rely on a binary signal from a pair of formulae for optimizing the parameters $\boldsymbol{\theta}$ of the ANN, given the nature of the feedback requested from the user (preferred formula between two options). In detail, a training data point for optimizing $\boldsymbol{\theta}$ is essentially a triplet that contains two formulae, f_1 and f_2, and a binary label $p \in \{-1, 1\}$. If the user deems f_1 to be more interpretable than f_2, then $p = -1$, otherwise $p = 1$. Given this data, we compute the interpretability scores $\psi_f(f_1)$ and $\psi_f(f_2)$ (one forward pass for each), and we compare them to p. Then, we use the Wasserstein loss function [136] to train the ANN:

$$\mathcal{L}\left(\psi_f\right) = p\left(\psi_f\left(f_1\right) - \psi_f\left(f_2\right)\right), \tag{8.1}$$

as done in [447]. The loss will be positive if the user and the estimator disagree on the relative interpretabilities of f_1 and f_2, whereas it will be negative in case of agreement. Note that this loss trains a *ranking* ANN rather than a regressor ANN. In other words, the ANN does not learn to estimate scores that have a specific meaning per se, rather, these scores are only meaningful in relative terms, i.e., to rank different models. As shown by Virgolin et al. [447], this training is effective and requires less annotation effort than estimating specific scores (i.e.,

a regression approach). Moreover, the employed model search algorithm, NSGA-II, relies for the most part on rankings rather than specific values. The only exception to this is the computation of the crowding distance, which is anyhow normalized, at each generation, using the minimum and maximum value for each objective encountered. We use Adam as optimizer [478], with a learning rate of 10^{-3} and a weight decay of 10^{-5}.

We argue that employing an ANN as an interpretability estimator for an AL strategy is a natural choice. This is because, using a given loss function, we can incrementally update the estimator model (i.e., the ANN weights). Gradients, which can be computed incrementally as feedback is received, facilitate this update process. In our setup, feedback involves comparing two formulae and selecting the better one. To address this, we utilize a Wasserstein-like loss function. Additionally, we seek a method to measure prediction uncertainty. If we were to use a different model, such as a DT, training the estimator with user feedback would convert the problem into a binary classification task, which does not support incremental gradient updates or provide a mechanism for measuring uncertainty.

ANN Warm-Up

As stated in Section 8.3, ML-PIE starts the search for a formula f using an initial interpretability estimator ψ_f^0. In practice, the initial estimator may have a substantial impact on what models are eventually discovered, as it drives the model search algorithm in a particular direction.

We therefore consider the effect of an optional *warm-up phase* prior to the actual start of ML-PIE. In other words, we consider the option of providing ML-PIE with a ψ_f^0 which is the result of an optimization performed on a dataset of triplets $\left(f_1^{(i)}, f_2^{(i)}, p^{(i)}\right)_i$, pre-collected and independent from the problem at hand—but compatible with it in terms of H and d. We elaborate below on choices to obtain the warm-up signal p. The warm-up phase needs not be too short (in terms of how many training triplets are given as data), as that will be ineffective, nor too long, as that will have the potential to lead to premature convergence to a sub-optimal notion of interpretability (w.r.t. the one of the user). With preliminary experiments, we find that using 20 triplets leads to reasonable results.

We consider the following two options to realize the warm-up.

ϕ-driven Warm Up. We rely on the ϕ interpretability estimator proposed by Virgolin et al. [445] for formulae, which we use to label 20 pairs of randomly generated formulae that use H and d variables. For each pair f_1, f_2 of randomly generated formulae, p is set to -1 if $\phi(f_1) \leq \phi f_2$ or to 1 otherwise. The ϕ estimator takes into account the number of components, the number of operations,

the amount of non-arithmetic operations, and the number of consecutive compositions of non-arithmetic operations in a formula. We remark that the encodings proposed in Section 8.4.2 use features that are more general than those used by ϕ, which is tailored for SR.

Data-driven Warm Up. As an alternative and potentially more general approach, we experiment also with a data-driven warm-up. In this case, we wish to use a dataset of "well-formed" and "elegant" models for the problem at hand, retrieved, e.g., from the internet. For the case study of SR, we particularly consider a subset of the dataset of the 100 equations from the Feynman lectures on physics [115, 211]. Specifically, we build 20 triplets f_1, f_2, p by extracting f_1 from the dataset, and building f_2 out of a random mutation to f_1 that makes the formula larger in terms of number of nodes for the respective tree. We make the assumption that, by its very construction, any f_2 is less interpretable than its respective f_1, and set the label p accordingly. Finally, to obtain a balanced dataset, we swap f_1 with f_2 and change the sign of the respective p for $50\,\%$ of the triplets.

We remark that this data-driven approach can be extended to other types of models than SR. For example, this approach could be used for the automatic inference of regular expressions, using a pool of human-written cases.

8.4.4 Query Building

The user is presented with a query of formulae $(f_1, f_2) \in F$ (F is the set of unique formulae available in the population of the current generation), where f_1 and f_2 have been picked by the AL algorithm. We consider two approaches to realize AL.

Random Sampling

We use *random sampling* without re-insertion, i.e., random f_1 and f_2 are extracted from F and the user will not see the same formula twice. While simple and fast to execute, this baseline approach is limited in that it makes no attempt at maximizing the information gain for training the ANN. However, random sampling can still be effective for very large input spaces (here, when the encoding has very high dimensionality).

Uncertainty Sampling

The second strategy is to use (the ANN) uncertainty sampling, which is a common strategy in AL and was found to be effective in [447]. To realize this, we assign to each formula a level of uncertainty using the ANN with dropout at prediction

time, as explained in Section 8.4.3. Namely, for any given formula in the set F of formulae seen during the current NSGA-II generation, $k = 10$ predictions are obtained from the ANN. These predictions are different from one another due to the ANN dropout. Then, the standard deviation of those predictions is taken as uncertainty. Finally, we pick the two formulae with the largest uncertainty among those in F.

Clearly, uncertainty sampling is more computationally heavy than random sampling, as it requires to evaluate the uncertainty for all the formulae in F for each query. In fact, this computation cannot be done beforehand, as the uncertainty might vary from query to query due to the update of ψ_f performed in between them, which can result in different uncertainties for the same formula.

8.5 Experimental Evaluation

We divide the experimental evaluation in three main phases, respectively aimed to (1) simulate how well the estimator ψ_f can be trained to mimic the user's preference (regardless of model search), (2) assess the impact of different user profiles (feedback speed) on the outcome of GP model search, and (3) deploy the most promising configuration of ML-PIE from the previous experiments on actual users and assess its effectiveness.

In the first experiment, we explore various methods to simulate the user's true notion of interpretability by considering different PHIs. We detail the PHIs used in the next subsection. This simulation is conducted across different encodings h, warm-up strategies, and AL sampling techniques. For the second experiment, we run ML-PIE with simulated users who follow different PHIs and exhibit various response patterns. In the third experiment, we engage 42 B.Sc., M.Sc., and Ph.D. students from engineering, computer science, and AI programs to provide feedback on two real-world datasets, which are described in Section 8.5.2.

All experimental phases are designed with the consideration that the proposed system will eventually involve actual human participants. We anticipate that participants may not be highly responsive and might be reluctant to provide extensive feedback. Therefore, it is reasonable to assume that participants are unlikely to spend several hours providing feedback to train the interpretability estimator. To address this, we aim to evaluate our system performance even with limited feedback. Consequently, experiments with the estimator are conducted using a small number of training pairs to reflect a realistic scenario. Additionally, experiments with GP are performed with a minimal number of generations and a small population size to ensure that the optimization process is efficient, reducing the likelihood of user fatigue and boredom.

Our code is available online[3]. We use Scikit-learn [332] and PyTorch [328] for the pre-processing of the datasets and the ANN training, respectively, and GenePro [441] and PyMOO [35] for the GP evolution. We remark that, unless otherwise specified, we set all the parameters to the values used originally used by Virgolin et al. [447] for ML-PIE.

8.5.1 Considered Proxies of Human Interpretability

For the first and second experiments, we utilize simulated users to explore and refine key aspects of ML-PIE before conducting the third experiment with actual users. These simulated users provide feedback based on a specified PHI, enabling us to generate a substantial amount of feedback data in a deterministic and consistent manner.

In our experiments, we evaluate ML-PIE using various interpretability measures, denoted as PHIs, to simulate different user preferences. Each PHI reflects a distinct approach to assessing model interpretability:

- *ℓ-PHI* evaluates formulae based solely on their size, specifically the number of nodes in the tree representation. Smaller formulae are deemed more interpretable, independent of the specific operations or primitives used.
- *ϕ-PHI* is based on the interpretability estimator ϕ from [445]. This measure considers multiple factors: the size of the formula, the number of operations, the number of non-arithmetic operations, and the length of consecutive non-arithmetic operation chains.
- *W-PHI* assigns weights to different primitives (functions, problem features, and constants) and evaluates formulae by summing the weights of the components present in the formula. Primitives with higher weights contribute more to the perceived complexity of the formula and thus lower its interpretability.
- *LW-PHI* is an extension of W-PHI, which assigns different weights based on the level of the primitive within the tree. Primitives at different tree depths have different impacts on interpretability.
- *NW-PHI* is the most detailed PHI, incorporating weights that depend on both the type of primitive and its specific position within the tree. This measure provides a nuanced evaluation of interpretability.

For W-PHI, LW-PHI, and NW-PHI, we simulate user preferences by sampling weights using the following formula:

$$w = -1 \times |\rho|, \text{ with } \rho \sim \mathcal{N}\left(0, \sigma^2\right). \tag{8.2}$$

[3] https://github.com/lurovi/ML-PIE

where the negative multiplication indicates that each included component decreases a formula interpretability. We choose σ^2 values based on the complexity of the primitives: smaller values for simple operators (e.g., $+, -, \times, \div$) and larger values for complex operators (e.g., $\ln$, $\max$), reflecting the intuitive understanding that simpler components are generally easier to interpret.

Note that, by construction, our PHIs are always negative. Moreover, we remark that the proposed PHIs do not necessarily capture some *true* notion of interpretability, e.g., using the model size has been often criticized in the interpretability literature, but they just act as proxies to simulate a possible user behavior.

8.5.2 Datasets Description

We run our GP-based experiments on two ML datasets for supervised regression tasks:

- *Boston housing* (Boston): dataset for discovering models that can predict prices of houses in different areas of Boston [144]. This dataset can also be used for assessing fairness in AI since it includes a feature regarding race. It contains 506 examples and 13 features;
- *Heating load* (Heating): dataset for discovering models that can predict heating load requirements of buildings. It contains 768 examples and 8 features [426, 425].

For each dataset, we perform three different $7{:}3$ random splits in training and test set. Features are normalized by using robust scaling. For each split, we run 10 different GP processes.

8.5.3 Training the Interpretability Estimator

We train the ANN using up to 150 training pairs, sampled according to each of the two AL approaches, from a training set consisting of 500 randomly generated trees. We choose 150 feedback rounds as a sufficiently large number for the expected number of feedback that users are willing to give. Each training pair is labeled according to a given PHI that simulates a possible interpretability notion of a generic user. For each feedback, we compute the Spearman footrule [394, 89] (a measure of how much two rankings mismatch) on a validation set consisting of 300 randomly generated trees. In this phase, we use six distinct variables $x_1, \ldots, x_6$ in the randomly-generated formulae. We repeat each experiment 10 times with different seeds, and we perform this type of experiment using every combination of the following parameters: (1) encoding h (Section 8.4.2); (2) PHI (Section 8.5.1); (3) AL method (Section 8.4.4); (4) warm-up strategy (Section 8.4.3).

Figure 8.4 shows the outcome of this experiment. Our findings are as follows:

- *Encoding.* Counts encoding generally performs best across various PHIs and AL methods, though the differences are not statistically significant. This encoding is notable for its low dimensionality and excellent scalability (see Section 8.4.2). It effectively handles simple PHIs, such as ℓ-PHI, but struggles with more complex PHIs, like NW-PHI (evidenced by the convergence of the footrule). In contrast, one-hot encoding is the most extensive encoding method we examine. It can lead to overfitting on simpler PHIs, such as ℓ-PHI (as indicated by the solid line for the random sampling strategy). However, one-hot encoding shows the potential to surpass counts encoding in handling complex PHIs like NW-PHI, though this requires a substantial amount of feedback. Given the limited training typically available in a human-in-the-loop setting, a smaller encoding is likely more suitable than a larger one.

- *PHI.* As anticipated, ℓ-PHI is the ground-truth that the estimator captures most easily, regardless of the encoding used, due to its simplicity. We observe that the footrule decreases even with more complex ground-truths like ϕ-PHI and NW-PHI. This suggests that a more general encoding might enable the estimator to approximate more complex PHIs, assuming the estimator receives sufficient feedback during training.

- *Active learning.* The results indicate that random sampling is generally as effective as, or better than, uncertainty-based sampling. In certain instances, such as W-PHI with Feynman warm-up and counts encoding, uncertainty-based sampling demonstrates a modest advantage over random sampling. We will explore this in more detail later in this section.

- *Warm-up.* In nearly all configurations, particularly with counts encoding, warm-up significantly reduces the footrule when no feedback has been provided yet. However, after approximately 20-25 feedback rounds, the advantage of warm-up diminishes, and its performance levels out with that of the no-warm-up approach. Additionally, using warm-up does not typically lead to premature convergence to a suboptimal estimator compared to not using warm-up. Overall, there is no significant difference observed between the two proposed warm-up strategies.

To provide a more detailed explanation of some previous observations, we conduct additional experiments where we measure the average uncertainty and the footrule on the validation set when the estimator is trained with significantly more feedback. Figure 8.5 displays the performance of the ANN when trained with up to 1000 training pairs. For this experiment, we use ϕ-PHI for labeling

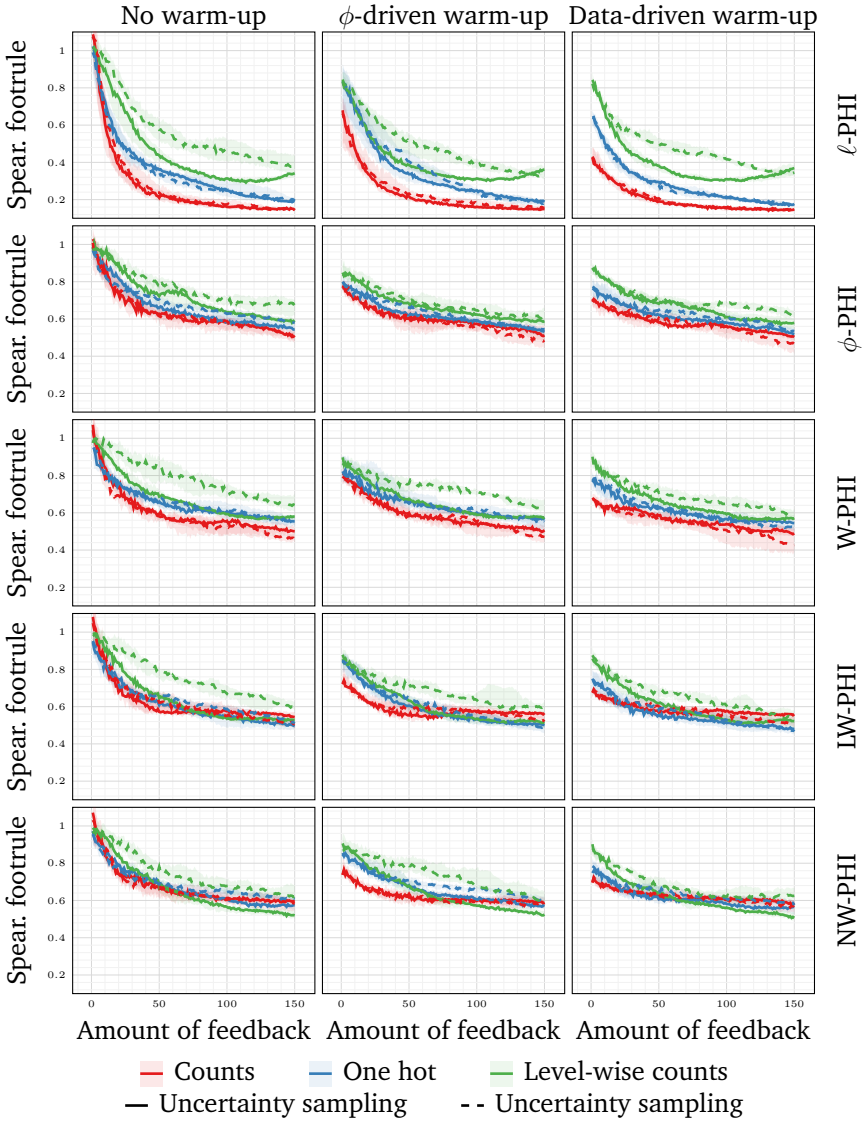


Figure 8.4: Median and inter-quartile range of Spearman footrule with increasing amount of feedback for different formulae encodings (color), sampling strategies (linetype), warm-up procedures (one per column), and different PHIs (one per row).

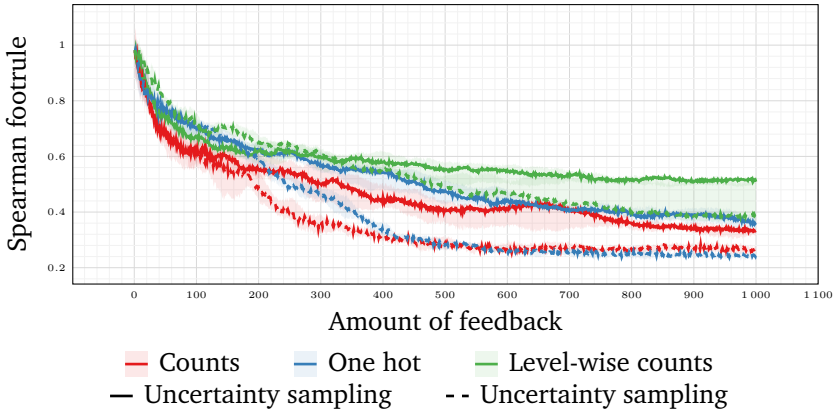


Figure 8.5: Median and inter-quartile range of spearman footrule with increasing amount of feedback for different formulae encodings (color) and sampling strategies (linetype) with no warm-up and ϕ-PHI.

the feedback and employ no warm-up. In this scenario, uncertainty sampling proves to be more effective than random sampling on average after 200–400 feedback rounds, regardless of the encoding used. This supports our hypothesis that uncertainty-based AL *can* surpass random sampling in effectiveness. However, depending on the complexity of the encoding, a substantial amount of feedback may be necessary before uncertainty sampling shows clear benefits.

Based on the aforementioned observations, we choose to adopt counts encoding, random sampling, and the ϕ-based warm-up for the following experiments. The decision regarding the warm-up strategy is grounded in statistical testing: Table 8.1 indicates whether using a warm-up strategy is significantly better than no warm-up when only a small amount of feedback—five rounds—is given. We select the ϕ warm-up because it yields smaller p-values compared to the data-driven warm-up across the different PHIs (except for ℓ, which is very simplistic). This makes the ϕ warm-up particularly suitable in scenarios where the user might be minimally responsive.

8.5.4 Integrating the Interpretability Estimator within GP

In this section we delve into the analysis of the full ML-PIE process, i.e., with multi-objective GP running while the user (in this section, still simulated) provides feedback to train the estimator of interpretability. As mentioned at the end of the previous section, and motivated by that section results, we adopt: counts encoding, random sampling, ϕ warm-up.

PHI	ϕ-driven vs. no warm-up	Data-driven vs. no warm-up
ℓ	0.097	0.004
ϕ	0.004	0.008
W	0.008	0.008
LW	0.004	0.027
NW	0.004	0.020

Table 8.1: p-values from Bonferroni-corrected Wilcoxon paired tests of the distribution of footrules (10 runs) at five feedback rounds (with random sampling).

In this experiment, we attempt to answer the following questions:

1. Do different user profiles induce different outcomes in terms of model accuracy? In other words, is GP search substantially influenced by the user's personal notion of interpretability?
2. Is the proposed approach robust w.r.t. different user behaviors in terms of engagement and response rate?

To this extent, we perform a two-fold experimental evaluation, as described below.

User Impact on Discovered Trade-Offs between Accuracy and Interpretability

We instantiate ML-PIE using a simulated user who responds at regular intervals of $5\,\mathrm{s}$, according to one of the PHIs. As PHIs, we consider a smaller but reasonable subset, comprising ℓ-PHI, ϕ-PHI, and NW-PHI. We consider ℓ-PHI and ϕ-PHI as they refer to common interpretability notions [445], but we also include NW-PHI, which is one of the most general notions we introduced. We repeat the model search 10 times for each combination of dataset, train-test split, and PHI, for a total of $10 \cdot 2 \cdot 3 \cdot 3 = 180$ runs. At the end of each run, we re-evaluate each solution found on the test set, to have an estimate of its generality.

From each Pareto front determined on the training set, we select formulae that correspond to different levels of interpretability and compare their performance in terms of accuracy. To facilitate this comparison, we rank the solutions on each front by their estimated interpretability and use the percentile τ to represent their position in the rank. Here, $\tau = 100$ indicates maximal interpretability with minimal accuracy, while $\tau = 0$ represents minimal interpretability with maximal accuracy. This ranking system enables us to compare models even when the underlying interpretability measure, i.e., PHI, varies.

Dataset	τ	R^2 Train	R^2 Test	Model
Boston	10	0.81	0.59	$\max(x_{12}, -0.15 - 4.39) - \max(x_9, x_5) + x_9 - x_3$
		0.77	0.78	$x_{12} - x_3 - x_{12} - \max(x_{12}, x_5)$
		0.75	0.72	$x_{12} - \max\left({x_1}^3 x_3 x_6, x_5 - x_{12} + x_{10}\right)$
	50	0.71	0.56	$x_{12} - x_5 - x_3 + x_{10}$
		0.72	0.68	$x_{12} - \max(x_{12} - \max(x_6, x_5), x_5)$
		0.72	0.63	$(x_4 + x_{10})\, x_5 - x_5 + x_{12}$
	90	0.57	0.54	$x_5 - x_8$
		0.27	0.29	$x_{11} - x_{11} + x_5 - x_8 - x_0$
		0.42	0.43	$x_{12} + \max(x_7, x_2 - x_7 - x_7)$
Heating	10	0.9	0.9	$x_6 - \ln(\lvert x_0 - \ln(\lvert x_4 \rvert)\rvert)$
		0.9	0.86	$x_6 + 4.61 x_2 + 2.33^3 (x_6 - x_3) - \max(x_2 x_2, x_5 + x_0) - \ln\left(\lvert \frac{x_4}{-3.75} \rvert\right)$
		0.93	0.94	$\ln(\lvert 0.14 - x_0 \rvert) - x_2 - \left(x_6 - x_0 - -2.22^3\right)\left({x_6}^3 + x_4\right)$
	50	0.82	0.79	$x_0 - x_4 - x_0 - \ln(\lvert x_3 \rvert)$
		0.9	0.86	$x_1 - x_4 - x_2 - x_4 - x_6$
		0.91	0.93	$x_3 - 4.5 - x_4 - x_6 + x_4 - x_0 - x_4$
	90	0.7	0.7	$x_6 - x_1 - x_4$
		0.8	0.73	$x_6 - x_3 - x_0 - x_2 - x_6$
		0.68	0.66	$x_0 - x_4 - x_0 - x_6 - x_0$

Table 8.2: Examples of models found by the bi-objective GP with a simulated user (NW-PHI), employing random sampling within AL and a ϕ-driven warm-up for the interpretability estimator. For each dataset, we report the results of three runs. We sample models according to their interpretability percentile (τ) in the Pareto front—higher τ meaning higher interpretability, and report the R^2 score of each model on both train and test sets.

Before discussing the quantitative results, in Table 8.2, we present examples of models along with their accuracy at different interpretability percentiles, τ, produced by GP. Qualitatively, it can be observed that as τ increases, the formulae become simpler and less accurate, which aligns with expectations.

Figure 8.6 shows the distributions of train and test accuracy—in terms of R^2 score—of the aforementioned models, for $\tau \in \{0, 25, 50, 75, 100\}$. We also report above each triplet of box-plots (corresponding to the three PHIs) the p-values resulting from a Kruskal-Wallis statistical test, with the null hypothesis that the considered distributions are not different from one another. We use a $\star$ to indicate cases with a p-value smaller than 0.01.

Figure 8.6 illustrates the interpretability-accuracy trade-off. Across all PHIs and both datasets, we observe that as τ increases, the (distribution of) R^2 decreases. This outcome is expected and serves as a sanity check to confirm that ML-PIE is functioning correctly. Assuming each PHI reflects a different user profile or subjective notion of interpretability, we can compare the R^2 distributions across different PHIs at each percentile τ to determine whether different PHIs lead to models with varying accuracy. Notably, based on the reported p-values, in 8 out of 10 cases for the training set and 7 out of 10 cases for the test set, the samples seem to originate from different distributions. This indicates that, in most cases, using a different PHI results in a different level of accuracy. In other words, the trade-off between interpretability and accuracy is highly dependent on the user's subjective notion of interpretability. This finding suggests that GP may need to be configured differently for different users. For example, if a user requires highly accurate models but has a notion of interpretability that makes it challenging to find such models, GP should be set to run with a larger budget.

Robustness of the Search Performance with Respect to User Engagement Profile

In the previous experiment, we simulated a constant rate of response of the user. In this section, we wish to assess whether changing this rate of response changes the behavior of GP. We thus repeat the previous experiment using two different user engagement profiles: a lazy-start profile and a lazy-end profile. For the lazy-start profile, we set the simulated user to provide feedback every $8\,\mathrm{s}$ for the first half of the model search (in terms of number of generations), and every $2\,\mathrm{s}$ for the remainder of the experiment; vice versa for the the lazy-end profile ($2\,\mathrm{s}$ for the first half, $8\,\mathrm{s}$ for the second half). Moreover, we also consider an additional configuration where we use the estimator learned with a constant rate of response of $5\,\mathrm{s}$, and use it directly from the start, i.e., as if it was an oracle. In this case, the user is not involved (i.e., the oracle is not updated). Similarly to the previous experiment, we repeat the experiment 10 times for each configuration, for a total of $10 \cdot 2 \cdot 3 \cdot 3 \cdot 3 = 540$ runs, and at the end of each run, we re-evaluate

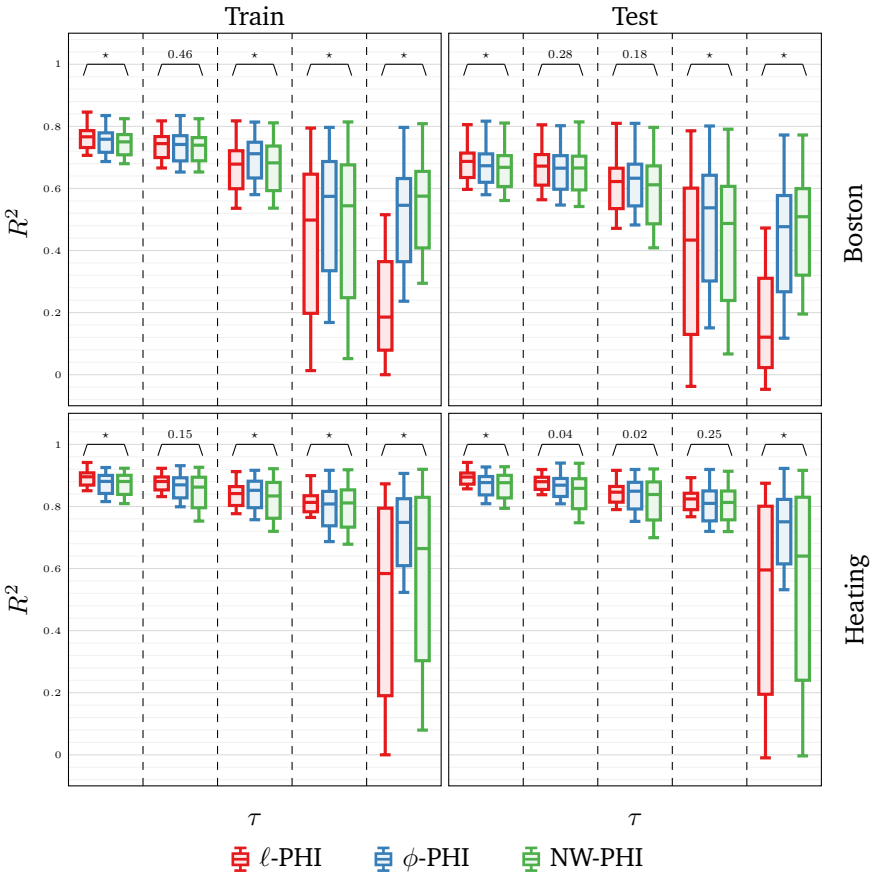


Figure 8.6: Distribution of R^2 for different levels of τ on training and test sets w.r.t. the different simulated user profiles. For each τ statistical significance is reported.

all solutions on the test set.

In the previous experiment, our focus was on determining whether different PHIs lead to variations in accuracy. Now, we shift our attention to examining whether the overall Pareto fronts differ when the user's response rate changes. To evaluate this, we use the HyperVolume (HV) indicator, which measures the "size of the space covered" by the solutions on the front [481]. Intuitively, a larger HV indicates better search performance, as it corresponds to a greater portion of the search space being covered.

Since we train ANNs to rank rather than to regress specific interpretability scores, the scores generated by ANNs trained on different runs (even for the same PHI) are not directly comparable. Therefore, to compute the HV, we calculate the actual PHI of each formula. This approach is reasonable because the estimator is intended to learn to rank models in a manner consistent with how PHI ranks them. Additionally, to facilitate comparison of HVs across different PHIs, we normalize the PHI scores using min-max normalization, ensuring that all values of PHI fall within the range of $[0, 1]$.

Figure 8.7 presents the distribution of hypervolumes (HVs) for each user profile, categorized by dataset (one per row), PHI (one per column), and whether the data is from the training or test set (shown on the x-axis). Additionally, above each set of four box plots, we include the p-values derived from a Kruskal-Wallis statistical test, which tests the null hypothesis that the distributions are equal.

The plots and p-values indicate that there are no significant differences in the hypervolumes (HVs) resulting from different user engagement profiles. This suggests that ML-PIE is robust to varying user behaviors, as long as a sufficient amount of feedback is provided during the model search process. Additionally, when considering the results obtained with the oracles, we observe that the model search process remains effective even when the interpretability objective changes over time.

8.5.5 Survey on Real Users

In the final experiment, we deploy ML-PIE with actual users. We involve 42 participants, consisting of B.Sc., M.Sc., and Ph.D. students from computer science and engineering programs at the University of Trieste, Italy. The participants are not offered any rewards for taking part in the experiment. The users are presented with a web interface where the landing page explains the task: to repeatedly select which of two formulae is more interpretable, first for the Boston dataset and then for the Heating dataset. The order in which the datasets are presented is randomized for each user. When a user clicks "start on dataset Boston/Heating", ML-PIE is initialized, and the interface shown in Figure 8.2 is displayed. ML-PIE operates with the same settings as in the previous experi-

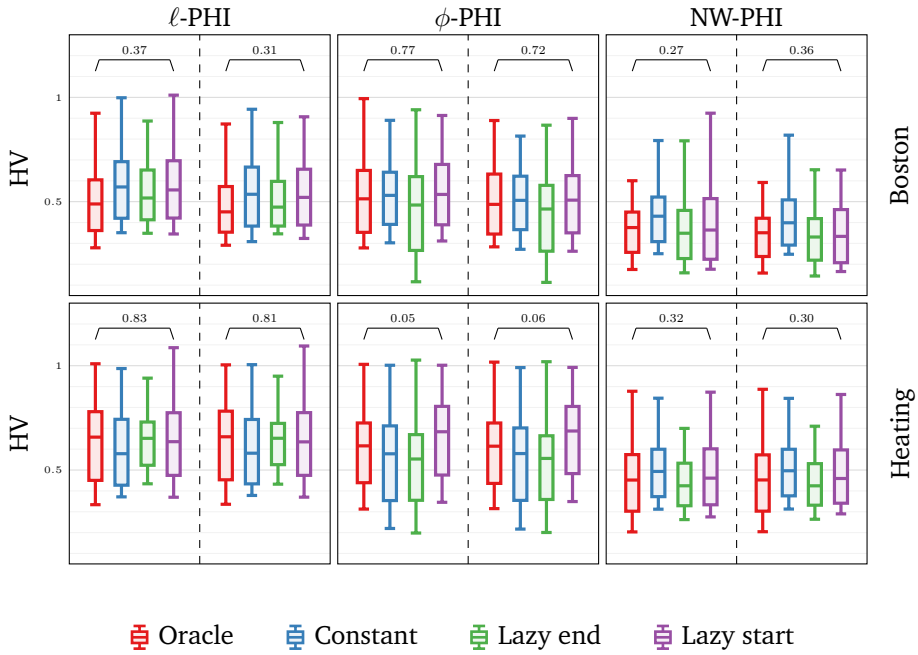


Figure 8.7: Hypervolumes distribution for different engagement profiles.

ments, resulting in approximately 10 minutes of runtime per dataset.

After ML-PIE terminates on a dataset, a further feedback page is presented. On this page, the user is presented with 4 pairs of formulae. The first pair contains the formula at $\tau = 25$ from the front obtained by the very ML-PIE run upon which the user had been giving feedback, to be confronted with another formula obtained in an off-line run that used ϕ as notion of interpretability, and that has the closest accuracy to the first formula (for the same dataset and train-test split). The other pairs are obtained in a similar manner, changing τ to 5, and/or ϕ to ℓ. We do not tell the user which formula was obtained with ML-PIE, and randomize whether the formula obtained with ML-PIE appears as first or second in the pair (blind test). In this comparison, besides indicating preference for the first or second formula, the user can indicate that they find both formulae to be equally (un)interpretable. Ultimately, our survey should capture whether users actually prefer the personalized approach that ML-PIE is intended to provide.

Figure 8.8 presents the results from the user survey. It shows that, on average, users tend to prefer models obtained from offline runs using ℓ or ϕ over those found by ML-PIE. This outcome is somewhat disappointing and could be

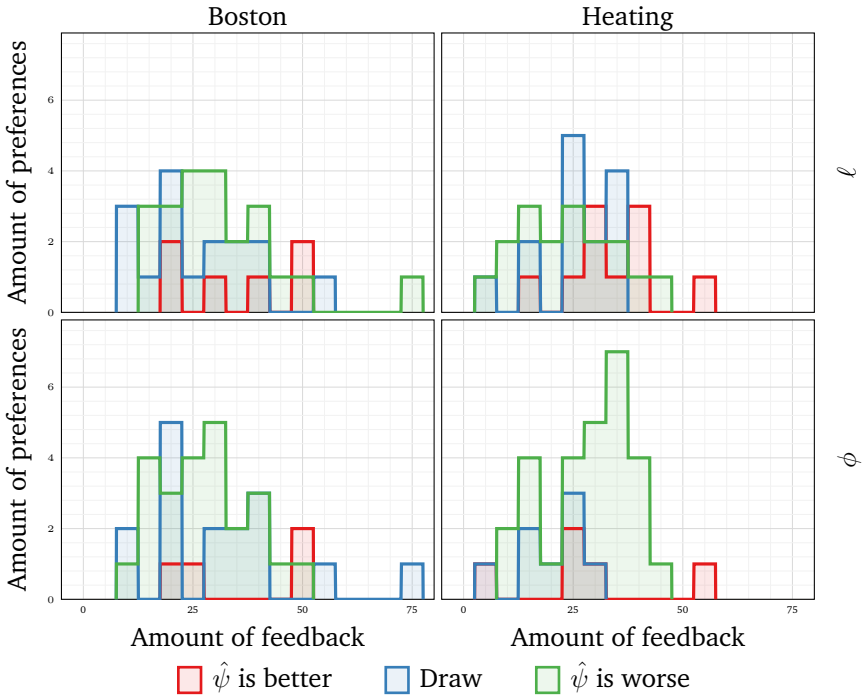


Figure 8.8: Amount of preferences for each model collected in the survey at the end of the user study w.r.t. the amount of feedback provided by the user during the GP run. We consider bins of size 5 for the x-axis.

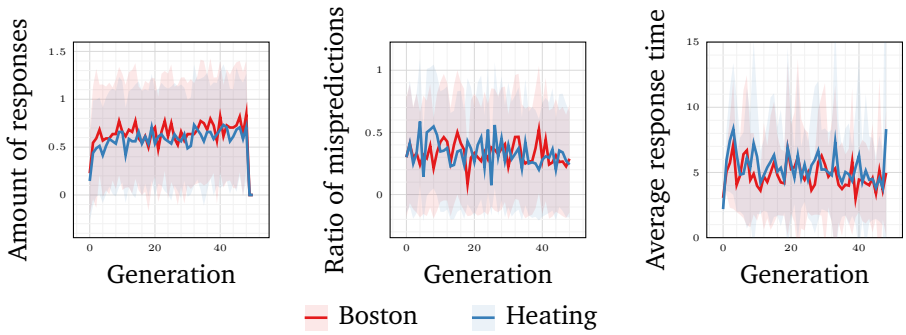


Figure 8.9: For each problem, mean and standard deviation range of the amount of responses, ratio of mispredictions the estimator makes in ranking the models (prior to the user's feedback signal), and average response time, along generations.

attributed to several factors: users may lack domain expertise, and their engagement in the feedback process may be minimal (no rewards are provided). Further investigation reveals that we barely achieve 50 feedback rounds in these experiments, as shown in Figure 8.9. Comparing this with Figure 8.4, it is evident that the estimator typically struggles to rank models accurately with fewer than 50 feedback rounds. Indeed, Figure 8.9 also shows that the amount of mispredictions—we take the relative ranking between the two formulae in the query as predicted by the estimator and compare it with the user's feedback—is relatively large (0.5 corresponds to coin flip).

To better understand this outcome, we examine the findings of [447]. In that study, users provided more feedback on average—about 100 rounds compared to 50 in our study—indicating higher engagement. Another key difference is that the encoding used by Virgolin et al. [447] was smaller and more specifically tailored for SR, relying on only four formula features. In contrast, our study employs counts encoding, which is more general. This suggests that while general encodings, like counts encoding, offer promising avenues for making ML-PIE more versatile and applicable across different problems, they might be too large to be effective in a human-in-the-loop setting. Consequently, the performance may be compromised if the encoding is not well-suited to the specific context.

However, there may be potential for improvement by developing more effective warm-up strategies. If such strategies could reduce the amount of feedback needed to achieve satisfactory results, it could enhance the effectiveness of ML-PIE even with general encodings.

8.6 Concluding Remarks

In this chapter, we explored various aspects to generalize *ML-PIE*, a human-in-the-loop approach for learning personalized interpretable models. We introduced three distinct representations with varying levels of complexity and expressiveness, which map GP trees with arbitrary functions and terminal sets to fixed-sized numerical embeddings. This extension enables the application of the original ML-PIE framework to problems beyond SR.

We investigated how effectively these representations allow an estimator of interpretability (in this case, an ANN) to rank models according to different PHIs from the literature, simulating different user profiles. Our findings indicated that while larger, more complex representations enable the estimator to better approximate more intricate PHIs, they require extensive training. Simpler representations, on the other hand, achieved reasonable results with less feedback but struggled to improve accuracy significantly with additional feedback.

Using GP as the model search algorithm, we simulated various user profiles to assess how they impact the models ML-PIE can discover, in terms of accuracy and the hypervolume of the Pareto fronts. We observed that different PHIs, representing diverse user preferences, resulted in varying model accuracies. However, ML-PIE demonstrated good robustness concerning the rate of feedback from the simulated users.

In the final experiment, real users (students) used ML-PIE and provided feedback in a blind survey, comparing models learned with ML-PIE to those learned with unpersonalized PHIs. This experiment yielded a negative result, suggesting that the use of a general but large encoding may necessitate excessive feedback for ML-PIE to be effective.

9

Interpretable Controllers with Graph-based Genetic Programming

In many high-risk scenarios, interpretability is essential for ensuring the safety and trustworthiness of systems. However, much of the current research depends on complex, highly parameterized models like ANNs that are difficult to understand. In this chapter, we address the challenge of policy search in continuous observation and action spaces. We employ two Graph-based Genetic Programming (GGP) techniques—Cartesian Genetic Programming (CGP) and Linear Genetic Programming (LGP)—to create effective yet interpretable control policies. Our experiments across eight continuous robotic control benchmarks demonstrate results that are competitive with state-of-the-art RL algorithms. Furthermore, we observe that GGP naturally tends to produce smaller, more interpretable graphs even when performance matches that of RL. Analyzing these graphs allows us to explain the derived policies, contributing to the development of trustworthy AI in continuous control applications.

This chapter is based on the following publication: **G. Nadizar**, E. Medvet, and D. G. Wilson. Naturally interpretable control policies via graph-based genetic programming. In *European Conference on Genetic Programming (Part of EvoStar)*, pages 73–89. Springer, 2024 [302].

9.1 Introduction

Over the past decade, ANNs have delivered remarkable results across a wide range of fields, often exceeding human performance in situations requiring quick decision-making, such as drone control [184], or in long-term strategic planning, like in strategy games [390]. This success has been accompanied by a proliferation of optimization techniques for ANNs [389], which have evolved to become sample-efficient, frequently robust, and even applicable in real-world scenarios [372].

However, ANNs have a significant drawback: their vast architectures, often comprising billions of parameters, are typically incomprehensible to human observers. Despite the introduction of various post-hoc explanation methods in recent years [135, 2, 348], ANN-based controllers continue to function as opaque models, where their inner workings can only be inferred through statistical analysis.

In contrast to this paradigm, there has been a resurgence of interest in creating *interpretable models*, which are models that can be understood "directly" without the need for additional proxies or post-hoc methods. Although there is no universally accepted definition of interpretability [235, 447, 303]—as we have vastly observed in Chapter 8, models that adhere to some interpretation standard are generally preferred over opaque models [367], as they enable humans to grasp their underlying logic directly. Unsurprisingly, interpretability is especially crucial in high-stakes applications, where transparency is a key component of trustworthiness [129].

In particular, the field of robotics is increasingly recognizing the demand for more transparent systems, as many people are hesitant to embrace autonomous robots in real-world environments, especially when their control algorithms are opaque. This need for transparency extends to the broader area of control systems, where the goal is to establish a control law that connects inputs to outputs in order to achieve a specific objective.

In this chapter, our goal is to discover efficient yet interpretable control laws by utilizing functional graphs, which are effective at capturing complex relationships between observations and actions while retaining interpretability, particularly when their size is contained. To explore the space of these graphs, we employ two GGP techniques, CGP and LGP, focusing optimization on performance while allowing interpretability to emerge naturally.

Our experiments, conducted on eight continuous control tasks from the Mujoco suite [422], demonstrate that the policies we obtain often match state-of-the-art performance while being inherently more interpretable, even without an explicit focus on interpretability. Additionally, the results achieved using GGP techniques appear more consistent and less dependent on hyper-parameter tun-

ing compared to those produced by RL.

9.2 Related Works

In recent years, the field of Explainable Reinforcement Learning (XRL) has gained significant momentum [348, 456], driven largely by the need to understand the reasoning behind decisions made by artificial agents intended for real-world deployment. Going a step further, considerable efforts have been directed toward developing policies that are inherently understandable, laying the groundwork for Interpretable Reinforcement Learning (IRL) [128]. In this context, numerous proposals for interpretable policies have emerged, ranging from symbolic policies to DTs, along with a variety of methods for generating them [213]. A popular approach within this domain is the use of EC techniques in conjunction with traditional RL methods [480]. Notably, GP and its derivatives are particularly well-suited for producing interpretable policies: as early as the 1990s, Koza and Rice [203] demonstrated how GP could be employed to search for a policy to control a robot.

Building on this foundational work, GP has frequently been applied to various control tasks, either through direct policy search or by imitation learning, where an agent learns by emulating the actions or behavior of an expert, i.e., a well-performing policy. For example, Verma et al. [437] distilled an ANN policy into an interpretable program for the Open Racing Car Simulator (TORCS), achieving interpretability and robustness albeit with a minor decrease in performance. However, while imitation learning partially mitigates the poor sample efficiency of GP [389], it is not always as effective as direct policy search. Specifically, Hein et al. [150] demonstrated in a model-based RL setting that symbolic policies learned through direct interaction with the world model outperform those obtained through SR to mimic a pre-trained ANN.

Instead, many studies have utilized GP for direct policy search, primarily focusing on scenarios with discrete action spaces. For instance, Custode and Iacca [71] and Ferigo et al. [108, 111] have evolved DTs by combining GP with RL and Quality-Diversity (QD) optimization, respectively, achieving both interpretable and effective policies for simple control tasks such as cart pole and mountain car. Video games have frequently served as test-beds due to the challenge of processing high-dimensional visual inputs from a full screen of pixels. In this area, significant effort has been directed toward the Atari benchmark [248], using approaches like co-evolved DTs [73] or Tangled Program Graphs (TPGs) [187], even developing single programs capable of solving multiple games [188, 189]. Related to our work, Wilson et al. [458] employed mixed-type CGP to discover simple and effective policies for playing Atari games. Notably, as in our approach,

the simplicity of these policies was not explicitly encouraged during evolution but emerged naturally due to the representation employed.

Moving closer to our study, fewer works have successfully addressed problems in continuous and multi-dimensional action spaces, as these problems necessitate discovering and leveraging interdependencies between outputs to achieve effective coordination. For example, CGP has been applied to continuous control problems; Wilson et al. [459] introduced a positional variant of CGP, where the positions of nodes were evolved, and tested it on nine benchmark problems, including three Mujoco environments. This study is closely related to ours, though we use the standard CGP variant across a broader range of environments and achieve generally better results. Videau et al. [439] also employed multi-tree GP for solving Mujoco environments and utilized LGP to facilitate information sharing among outputs. This work is related to ours, although they focus on bi-objective optimization with an explicit reward for policy simplicity, whereas we allow simplicity to emerge naturally from evolution. Additionally, we examine a larger set of environments. More recently, LGP has been successfully used with the Laikago robot to develop robust and understandable policies, represented as transparent computer programs [190]. Lastly, Amaral et al. [7] demonstrated the potential of Symbiotic Bid-based GP combined with TPGs for a continuous control locomotion task.

9.3 Graph-based Genetic Programming for Continuous Control

We concentrate on the domain of continuous control, with the aim of finding an *interpretable* yet effective controller, whose actions will steer the system towards the achievement of a certain goal. Namely, we consider those tasks where both the observations and the possible actions are real-valued and multivariate, in a discrete-time simulated environment.

Formally, we define an environment as a Markov Decision Process (MDP), i.e., a tuple $(\mathcal{S}, \mathcal{A}, p, r)$, where $\mathcal{S} \subseteq \mathbb{R}^n$ and $\mathcal{A} \subseteq \mathbb{R}^m$ are the state and action spaces, respectively, $p : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the function describing the dynamics of the environment, and $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function [349]. We consider partial observability, thus we introduce an observation space $\mathcal{O} \subseteq \mathbb{R}^q$, with $q \leq n$, and an observation function $\phi : \mathcal{S} \to \mathcal{O}$ mapping states to the corresponding observations. At each simulation time step t, the environment is in a state $\boldsymbol{s}_t \in \mathcal{S}$, the agent observes a subset of the current state, $\phi(\boldsymbol{s}_t) = \boldsymbol{o}_t \in \mathcal{O}$, and takes an action $\boldsymbol{a}_t \in \mathcal{A}$ according to a *policy* $\pi : \mathcal{O} \to \mathcal{A}$, which results in the system changing into state $\boldsymbol{s}_{t+1}$ with probability $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$ and the agent receiving a reward $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$. The objective consists in finding a policy

which maximizes the cumulative reward obtainable in a simulated episode of duration T, $R_T(\pi) = \sum_{t=0}^{T} r(\boldsymbol{s}_t, \pi(\boldsymbol{o}_t))$, starting from an initial state $\boldsymbol{s}_0$. While we consider the formalization of an MDP, in this study we use optimization methods based only on the total episode reward $R_T(\pi)$. Unifying the terminology with the previous paragraph, π is the controller we search for, which determines the actions in the environment.

In our case, π is a graph which represents a multi-variate function. This family of policies can be inherently interpretable because each graph results from the high-level composition of simple functions, e.g., $+$, $-$, or $\sin$, and is constrained in size. We rely on two flavors of GGP for searching the space of graphs, namely CGP and LGP (see Section 2.3.2 for a description of both techniques). We define the fitness of a policy graph π as the aforementioned cumulative reward, $f(\pi) \coloneqq R_T(\pi)$, which evolution maximizes. For the evolutionary optimization we rely on a standard GA (Algorithm 2.1) with the representation dependent implementations of `init()` and `variate()` detailed in Section 2.3.2.

9.4 Experimental Evaluation

In our experimental evaluation we address the following question: *"can GGP yield effective yet interpretable policies for continuous control tasks?"* To this end, we perform several optimizations for both CGP and LGP on 8 continuous control tasks from the Mujoco suite. To have meaningful effectiveness baselines, we compare the performance of the resulting graph policies with those obtained with two state-of-the-art RL algorithms which represent policies using ANNs. Moreover, we measure the complexity of the graphs obtained by GP, and analyze example policies in detail to appraise their interpretability.

We open-source our code and our experimental results[1].

9.4.1 Continuous Control Benchmark Tasks

We employ 8 benchmark continuous control tasks from the Mujoco suite [422]. Namely, we consider environments of growing complexity in terms of input and output space dimensionality, as we summarize in Table 9.1.

Specifically, we consider two balancing tasks, *inverted pendulum* and *inverted double pendulum*, where the controller is rewarded for preventing the pendulum from falling (and also penalized for excessive movements for the inverted double pendulum). We also incorporate a target-aiming task, *reacher*, where a robotic arm has to reach an object and is rewarded for getting as close to the object as possible and penalized for excessive movements. Last, we include 5

[1] `https://github.com/giorgia-nadizar/cgpax`

Environment	Observation $\mathcal{O}$	Action $\mathcal{A}$
Inverted pendulum	$\mathbb{R}^4$	$[-3, 3]$
Inverted double pendulum	$\mathbb{R}^8$	$[-1, 1]$
Reacher	$\mathbb{R}^{11}$	$[-1, 1]^2$
Swimmer	$\mathbb{R}^8$	$[-1, 1]^2$
Hopper	$\mathbb{R}^{11}$	$[-1, 1]^3$
Walker2d	$\mathbb{R}^{17}$	$[-1, 1]^6$
Half cheetah	$\mathbb{R}^{18}$	$[-1, 1]^6$
Ant	$\mathbb{R}^{27}$	$[-1, 1]^8$

Table 9.1: Observation and action space sizes for the considered problems.

locomotion tasks, *swimmer*, *hopper*, *walker2d*, *half cheetah*, and *ant*, which all use a positive reward for the distance covered and a penalty term for excessive movements. Among the locomotion problems, hopper, walker2d, and ant can be unstable; they therefore have an additional reward term for maintaining their balance throughout the simulation. For these problems, if the agent falls to the ground, the episode is terminated earlier.

For all tasks, we use the Brax [120] implementation which leverages parallelism on GPU accelerators with JAX [43]. We use the v1 simulation engine of Brax and set all parameters to their default values. We run all episodes to a duration $T = 1000$ time steps.

9.4.2 Reinforcement Learning Baselines

As baselines for comparing the performance of the graph policies, we use two state-of-the-art RL algorithms for the optimization of ANNs, Proximal Policy Optimization (PPO) and Soft Actor Critic (SAC).

PPO [379] is an on-policy RL algorithm that optimizes agent policies within a "trust region", balancing the exploration-exploitation trade-off. It achieves this by optimizing a first-order approximation of the expected reward while ensuring that policy updates remain close to the actual values.

SAC [139] is an off-policy RL algorithm that balances the maximization of two objectives: the expected cumulative reward and the policy entropy. By simultaneously promoting exploration through entropy maximization and exploitation through return maximization, SAC ensures robust learning in complex environments with continuous action spaces.

These two algorithms are considered state-of-the-art for deep RL and are the default algorithms for the Brax library. We use the Brax library implementations for both PPO and SAC.

9.4.3 Parameter Settings

For the GP evolutionary loop we set $n_{\text{pop}} = 100$, $n_{\text{off}} = 90$, $n_{\text{elite}} = 10$, and $n_{\text{tour}} = 3$. Concerning the amount of evaluations performed n_{evals}, instead, we have a different value for each problem, depending on how difficult the optimization task is. Namely, for inverted pendulum we set $n_{\text{evals}} = 1000$, for half-cheetah we set $n_{\text{evals}} = 50\,000$, for inverted double pendulum, reacher, and swimmer we set $n_{\text{evals}} = 100\,000$, for hopper and walker2d we set $n_{\text{evals}} = 150\,000$, and for ant we set $n_{\text{evals}} = 250\,000$. Regarding the representation specific parameters, we set $n_{\text{nodes}} = 50$, $p_i = p_f = 0.1$, and $p_o = 0.3$ for CGP, while we set $n_{\text{reg}} = n_{\text{in}} + n_{\text{out}} + 5$, $n_{\text{lines}} = 15$, $p_a = 0.3$, and $p_f = p_i = 0.1$ for LGP.

For both CGP and LGP we consider the following function set $H = \{\bullet + \bullet, \bullet - \bullet, \bullet \times \bullet, \bullet \div^* \bullet, |\bullet|, \exp \bullet, \sin \bullet, \cos \bullet, \log^* \bullet, \bullet < \bullet, \bullet > \bullet\}$, where $\bullet$ represents an operand, and operators marked with * are protected. The last two functions are Boolean functions, where the output is 1 if the condition is satisfied and 0 otherwise. Moreover, we add two constant inputs to each observation, namely $\{0.1, 1\}$, thus increasing all observation spaces dimensionalities displayed in Table 9.1 by 2.

For the RL algorithms, we optimize an MLP with 4 layers of size 32 with Sigmoid Linear Unit (SiLU) activation for PPO, and an MLP of 2 layers of size 256 with ReLU activation for SAC, following the default provided in Brax. Concerning the algorithm specific hyper-parameters we apply the default ones from the Brax implementations of both PPO and SAC with a few minor variations (we report the full parameter list in our code[2]).

In all cases, we repeat the optimization for 10 independent times to ensure results consistency. Furthermore, for the GP techniques we evaluate each individual on 5 distinct episodes and consider the median reward across these episodes as fitness, in order to make the evolutionary search more stable. For the same reason, we also re-evaluate the elite individuals upon merging them into the offspring population.

Last, when analyzing results in a comparative manner we perform a Mann-Whitney U test between pairs of distributions, considering equivalence between the two as null hypothesis. When performing multiple pairwise comparisons, we apply the Bonferroni correction, thus dividing the significance level $\alpha = 0.05$ by the number of pairwise comparisons computed.

[2] https://github.com/giorgia-nadizar/cgpax/blob/main/rl_run.py

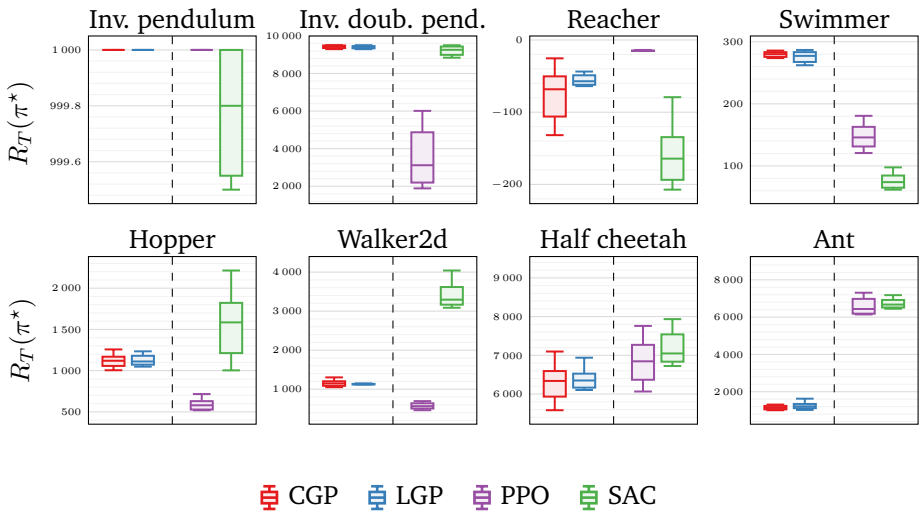


Figure 9.1: Box plots of the cumulative reward $R_T(\pi^\star)$ collected by the best policy discovered at the end of optimization $\pi^\star$ in an episode of duration $T = 1000$ time steps.

9.5 Results and Discussion

9.5.1 Performance Results

We report the results in terms of performance in Figure 9.1. We show the distribution across 10 runs of the cumulative reward $R_T(\pi^\star)$ obtained by the best policy $\pi^\star$ discovered at the end of the optimization on episodes of $T = 1000$ time steps. We divide our data by environment and by optimization technique to ease the visual comparison of results. We also show the p-values resulting from the pairwise Mann-Whitney U tests performed in Table 9.2, again dividing data by environment.

The first observation we can make is regarding the comparison between the two GGP techniques employed: in all the environments considered, they achieve similar results, with no statistically significant differences observed between the distributions. Additionally, the results for both CGP and LGP are generally consistent, unlike the results from the two RL techniques, where the distributions are more spread out and show greater variability between the PPO and SAC algorithms. This indicates that GGP offers more consistency than RL, which is a desirable property for achieving robust results. We speculate that this greater consistency may be partly due to the high sensitivity of RL algorithms to their

	PPO	SAC
CGP	0.168	0.078
LGP	0.168	0.078

(a) Inv. pendulum

	PPO	SAC
CGP	**0.005**	0.571
LGP	**0.006**	0.623

(b) Inv. doub. pend.

	PPO	SAC
CGP	$\approx$**0**	**0.003**
LGP	$\approx$**0**	**0.003**

(c) Reacher

	PPO	SAC
CGP	$\approx$**0**	$\approx$**0**
LGP	$\approx$**0**	$\approx$**0**

(d) Swimmer

	PPO	SAC
CGP	$\approx$**0**	0.021
LGP	$\approx$**0**	0.017

(e) Hopper

	PPO	SAC
CGP	$\approx$**0**	$\approx$**0**
LGP	$\approx$**0**	$\approx$**0**

(f) Walker2d

	PPO	SAC
CGP	0.064	**0.003**
LGP	0.031	**0.001**

(g) Half cheetah

	PPO	SAC
CGP	$\approx$**0**	$\approx$**0**
LGP	$\approx$**0**	$\approx$**0**

(h) Ant

Table 9.2: p-values resulting from pairwise Mann-Whitney U tests comparing, for each environment, the algorithm on the row with that on the column with the null hypothesis of equivalence of the distributions. We write ≈ 0 for all cells with $p < 0.001$. We consider a significance level of $\alpha = 0.05/5 = 0.01$ (and highlight all significant values in bold) according to the Bonferroni correction, as we performed 5 pairwise comparisons per environment: the ones reported in the table plus the comparison between CGP and LGP (which never yielded statistically significant differences).

hyper-parameters.

For a more detailed comparative analysis, we revisit the first part of our question, which concerns the effectiveness of the graph-based policies discovered. In this context, a policy is considered effective if it performs comparably to the examined RL baselines, without being significantly worse. Given the reduction in the number of parameters and the resulting gain in interpretability, policies that achieve results similar to, though not necessarily better than, state-of-the-art policies are deemed effective.

Among the environments studied, graph-based policies are not significantly worse than either RL policy in $50\,\%$ of the cases (inverted pendulum, inverted double pendulum, swimmer, and hopper). Additionally, they perform at least as well as one of the RL policies in three more cases (reacher, walker2d, and half cheetah). The only environment where graph-based policies fail to match either of the RL algorithms is the ant, likely due to the higher dimensionality of both the

action and observation spaces, as well as the known challenge of GGP in finding effective mappings in high-dimensional spaces. Overall, we can conclude that both CGP and LGP policies are generally effective.

Interestingly, in some instances, GGP methods significantly outperform RL. Specifically, in four environments (inverted double pendulum, reacher, hopper, and walker2d), both CGP and LGP notably exceed the performance of one of the two RL algorithms. In the swimmer environment, both GP methods outperform both RL algorithms. It is possible that some of these results could be due to sub-optimal parameter tuning for the underperforming RL algorithms, which might achieve better performance with more extensive fine-tuning. Nonetheless, this observation further supports our earlier point about the robustness of GP results w.r.t. hyper-parameter tuning, in contrast to RL, which is highly sensitive to changes in hyper-parameters [151].

To gain a deeper understanding of the evolutionary process behind the GP policy search, we examine the progression of the fitness of the best individual in the population, specifically the cumulative reward $R_T(\pi^\star)$, at each iteration, as shown in Figure 9.2. For all the reported plots, we observe that evolution tends to reach a plateau, where further generations are unlikely to yield significant improvements in fitness. Additionally, most plots exhibit an initial phase of steep fitness growth, followed by a longer period of minor improvements. This pattern highlights how evolution generally converges relatively quickly—a characteristic with both positive and negative implications. On the positive side, for environments where the results are satisfactory (i.e., comparable to state-of-the-art performance), this rapid convergence translates to reduced computational costs. On the other hand, for tasks where GP has not matched the performance of RL, this rapid convergence indicates that evolution may become trapped in a difficult to escape local optimum, which limits overall performance.

To further investigate the reasons behind premature convergence, we analyze videos of the behaviors of the graph policies and examine the reward models for the hopper, walker2d, and ant environments. From the visual analysis, we observe that the graph policies cause the agents to move, but the movements are slow and overly cautious. This observation aligns with the reward model of the environments, where: (1) agents are rewarded for maintaining stability, and (2) episodes are terminated early if an agent falls, thus preventing the collection of additional rewards. We speculate that this reward model creates a large local optimum for GGP, where innovative policies that might require fine-tuning to achieve stability are overshadowed by evolutionary pressure.

In line with this hypothesis, we conduct experiments by disabling the early termination of episodes when the agent falls. However, the results do not show significant improvements, suggesting that further investigation is needed to fully understand this phenomenon. Interestingly, we discover some physically unre-

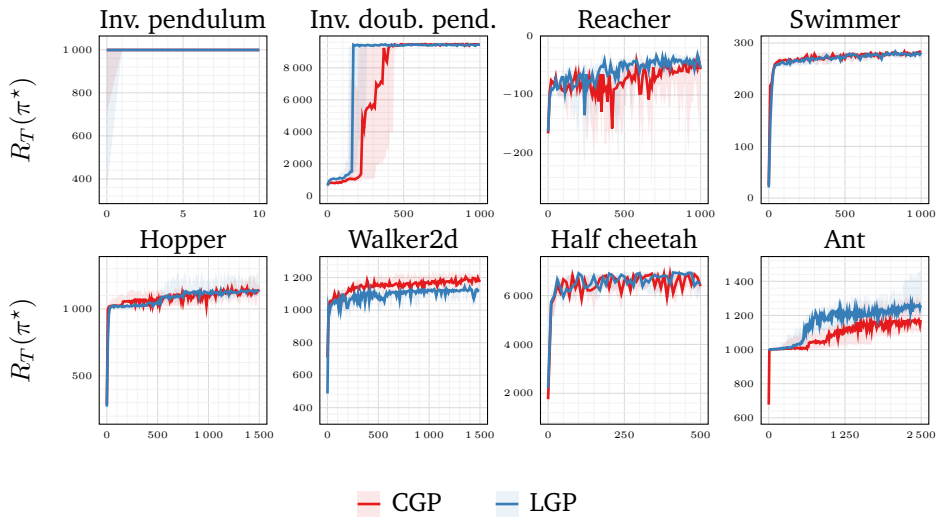


Figure 9.2: Progression of the cumulative reward $R_T(\pi^\star)$ collected by the best policy in the population $\pi^\star$ at each generation in an episode of duration $T = 1000$ time steps; median and inter-quartile range across 10 independent runs.

alistic policies for the hopper, which could achieve a reward as high as $30\,000$ by causing the agent to fly[3]. These policies clearly exploit certain instabilities within the Brax simulator [120]. Despite their unrealistic nature, being transparent models, these policies could be thoroughly analyzed to address and correct the identified issues in the simulator.

9.5.2 Interpretability of Resulting Policies

To evaluate the interpretability of a graph policy π, we consider $\rho_c(\pi)$, which we define as the fraction of complexity employed w.r.t. the available one. In practical terms, for CGP, ρ_c is the fraction between the amount of nodes in the policy graph and the maximum possible nodes (50 in our case), whereas for LGP, ρ_c derives from the amount of program lines that contribute to the output computation divided by the total lines of the program (15 in our case).

Clearly, this serves as a *proxy* for interpretability, given that (1) there is no universally accepted definition of interpretability, and (2) interpretability is inherently a subjective concept [447]. Yet, given the results obtained in Chapter 8 on how most users' subjective preferences strongly correlate with models sizes

[3]We show a video at `https://giorgia-nadizar.github.io/cgpax/hopper_cgp_flying`.

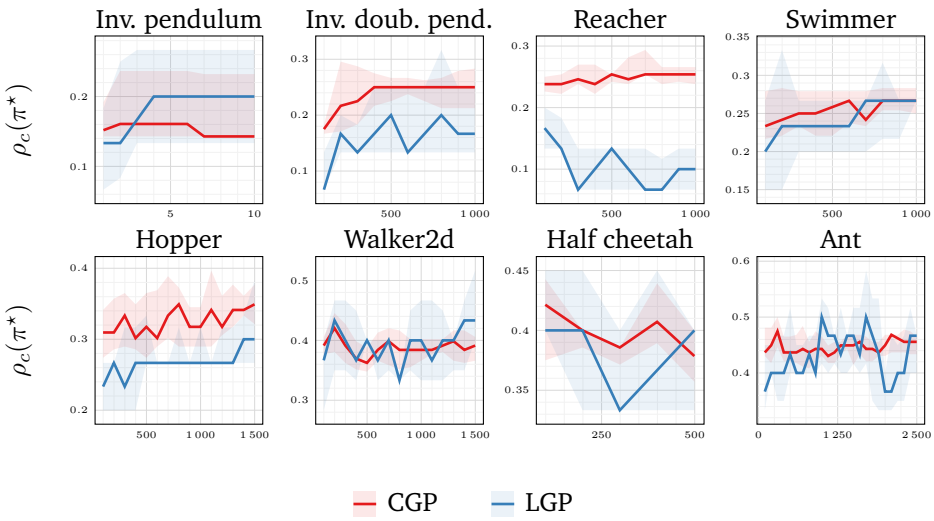


Figure 9.3: Fraction of complexity employed $\rho_c(\pi^\star)$ for the best performing policy graph in the population $\pi^\star$; median and inter-quartile range across 10 independent runs.

we deem this approximation acceptable for this case study. In addition, to ensure the robustness of our results, we repeat the analysis using two additional measures of interpretability—the number of edges in the policy graph and the formula ϕ from [445] (already used in Chapter 8)—obtaining similar results to those presented.

In Figure 9.3, we illustrate the progression of $\rho_c(\pi^\star)$, the fraction of complexity utilized by the best policy graph in the population $\pi^\star$, presented as median and inter-quartile range across the 10 runs, categorized by environment and GP technique. From the plots, it is evident that in all scenarios, the obtained policies use only a relatively small fraction of the available complexity, with ρ_c generally remaining below one-third throughout the evolution process. Additionally, there is no clear upward trend in ρ_c over generations: while some plots show a slight increase, most remain roughly constant or even decrease. This observation is particularly noteworthy when compared to Figure 9.2. It indicates that a significant increase in fitness does not necessarily lead to a more complex policy, meaning that improved performance does not come at the expense of interpretability.

Notably, these results naturally arise from the inherent representations of CGP and LGP [275, 117], without any explicit promotion of interpretability. This observation confirms the inherent tendency of these GGP techniques to produce

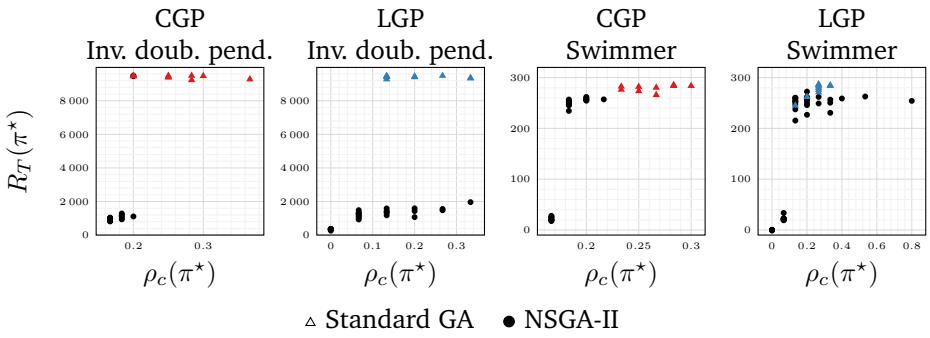


Figure 9.4: Cumulative reward $R_T(\pi^\star)$ vs. fraction of complexity $\rho_c(\pi^\star)$ for each policy $\pi^\star$ in each pareto front at the end of evolution with NSGA-II. Policies found with the standard GA added for reference.

small and interpretable graphs.

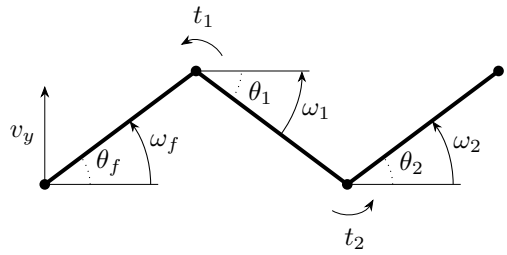
For completeness, we also conduct optimization in a bi-objective setting, focusing on both effectiveness and interpretability, similar to the approach in [439]. This involves using NSGA-II [86] (Algorithm 2.3) instead of a standard GA. We perform 10 runs for each scenario, though this time we only consider two environments: inverted double pendulum and swimmer.

We present the results in Figure 9.4, where we plot the cumulative reward $R_T(\pi^\star)$ versus the fraction of complexity $\rho_c(\pi^\star)$ for all policies $\pi^\star$ in each of the final Pareto fronts. For context, we also include the policies obtained with the standard GA. From this figure, we observe that NSGA-II is able to discover simpler policies compared to the standard GA. However, it often fails to identify slightly larger but more effective policies due to its tendency to prioritize the easiest objective [236]. Consequently, we conclude that for these GP techniques, where interpretability tends to emerge naturally, maintaining a single objective in the search process is generally advisable.

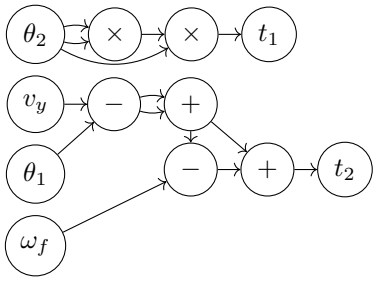
To conclude our study on the interpretability of the obtained policies, we analyze two policies for the swimmer environment. We present a schematic representation of the environment [66], along with the two policies and their corresponding equations, in Figure 9.5[4].

From Figures 9.5b and 9.5c, we can immediately appreciate the simplicity and transparency of the policies, which align well with our interpretability objective. While fully understanding a dynamical system typically requires examining all governing equations, such as those from the simulator, the reported formulae

[4]We provide videos of these policies in action at `https://giorgia-nadizar.github.io/cgpax/swimmer_cgp` and `https://giorgia-nadizar.github.io/cgpax/swimmer_lgp`.



(a) Swimmer schematic.




```
def controller(inputs, r):
  # inputs to registers
  r[0:10] = inputs
  # perform computation
  r[13] = r[1] - r[5]
  r[16] = r[16] - r[7]
  r[15] = r[13] - r[6]
  r[15] = r[15] + r[2]
  r[16] = r[16] - r[1]
  # registers to output
  return r[-2:]
```

$$t_1 = \theta_2^3$$
$$t_2 = 4(v_y - \theta_1) - \theta_f$$

(b) CGP solution.

$$t_1 = \theta_1 - \omega_f - \omega_1 + \theta_2$$
$$t_2 = -\omega_2 - \theta_1$$

(c) LGP solution.

Figure 9.5: Schematic representation (9.5a) of the swimmer environment with the observed variables $\theta_f, \theta_1, \theta_2, \omega_f, \omega_1, \omega_2, v_y$ and the control variables t_1, t_2, and the policies found with CGP (9.5b) and LGP (9.5c), shown in their original form and as formulae.

still provide valuable insights. Specifically, we observe that each torque control value depends on the state of the opposite side of the agent, whether in terms of angles or angular and linear velocities. This characteristic, which might be obscured in an opaque policy, is crucial for achieving coordinated movement and likely plays a significant role in the effectiveness at the locomotion task.

9.6 Concluding Remarks

In this chapter, we utilized two GGP techniques, CGP and LGP, to tackle various continuous control problems with the aim of deriving effective yet interpretable control policies. Our experiments across eight Mujoco environments revealed that the graph-based policies not only achieved state-of-the-art performance on several tasks but were also sufficiently simple to be observed and directly understood. We confirmed the inherent tendency of both CGP and LGP to produce simple graphs, even without explicit incentives for interpretability. Namely, we found that single-objective optimization generally performed better in striking a balance between performance and simplicity compared to bi-objective optimization.

10

Interpretable Graph Controllers via Quality-Diversity

Interpretable control policies represented in graph format can be obtained with GGP, but it encounters issues like local optima and solution fragility that reduce its effectiveness. QD has successfully tackled similar problems, particularly in ANN optimization. This chapter presents a general Graph Quality-Diversity (G-QD) framework designed to improve GGP performance through QD optimization, leading to a range of interpretable, effective, and robust policies. By employing CGP as the GGP method and MAP-Elites (ME) as the QD algorithm, we integrate both behavioral and structural graph descriptors. Our experiments, conducted on two navigation and two locomotion continuous control tasks, show that our framework generates a variety of effective policies with diverse behaviors and structures, outperforming traditional search methods. Additionally, the solutions derived are more interpretable, providing better insights into control tasks, and demonstrate resilience to issues such as sensor malfunctions.

This chapter is based on the following publication: **G. Nadizar**, E. Medvet, and D. Wilson. Searching for a Diversity of Interpretable Graph Control Policies. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 933–941, 2024 [301].

10.1 Introduction

Policy search and RL are gaining attention for their potential in critical fields like autonomous driving [12], power systems [479], and traffic management [228]. Typically, these control policies are modeled using ANNs, a complex, highly parameterized opaque model that often necessitates statistical post-hoc explanations for interpretation. This raises concerns about the applicability of such models, as stakeholders in critical domains generally demand control models that are explainable [367].

GP provides an alternative approach to policy representation by composing high-level functions throughout the optimization process [202]. A recent survey of the expanding IRL field reveals that several GP methods are being employed to develop interpretable policies that can be competitive with deep RL in certain situations [128]. Nonetheless, GP often prematurely converges to policies that underperform compared to their deep RL equivalents, suggesting the presence of policies trapped in local optima [189]—much like we have observed in Chapter 9. Additionally, research into these interpretable policies shows that they may evolve to rely on specific features or conditional logic, making them vulnerable to environmental changes [458].

To address the issues of premature convergence to local optima and reduced robustness, we investigate the use of QD to explore a diversity of policies. We propose a framework called G-QD that integrates QD with GGP. This framework incorporates graph structure descriptors to facilitate the search for a variety of functional structures, alongside utilizing the behavior descriptors of the evolved policies. We apply the G-QD framework to continuous control tasks, a particularly challenging area. Our study evaluates the quality, diversity, interpretability, and robustness of the discovered policies, demonstrating that G-QD positively influences all these aspects compared to a baseline GA.

10.2 Related Works

The growing necessity to understand the decisions made by artificial agents in real-world applications has led to increased interest in RL methods that generate inherently interpretable policies, thus advancing the field of IRL [128]. Within this framework, various policy representations and optimization techniques have been developed, a lot of which encompassing GP, as we have thoroughly examined in Section 9.2.

However, despite such significant efforts in GP policy search, the results have often not matched the performance of state-of-the-art RL baselines. A common issue is the limited expressivity of some GP representations, which fall short of

the complexity provided by large opaque models. Additionally, GP may underperform compared to other RL methods due to insufficient exploration and a tendency to get stuck in local optima. To address these limitations, Ferigo et al. [108, 111] proposed using QD optimization to evolve interpretable DTs for simple discrete action problems. Their method employs a single behavior descriptor, specifically the entropy across the action space, and a single policy interpretability descriptor, which is the depth of the DT. Our approach aligns with theirs, with two key distinctions: (1) we address more complex continuous actions problems using a different GP technique, and (2) we explore multiple combinations of higher-dimensional behavior and policy structural descriptors within a single archive and across multiple archives.

The concept of using multiple archives to store solutions is well-established in the QD literature. Many studies have combined Novelty Search (NS) with multiple archives to enhance exploration. For example, Doncieux and Coninx [92] utilized this approach to automatically generate new archives for open-ended evolution. Similarly, Morel et al. [287] and Marrero et al. [252] used multiple archives to improve exploration in a grasping task and to generate instances for a Knapsack problem, respectively. A closely related study is that of Cazenille [58], who employed multiple archives within the ME variant AURORA [68] to achieve greater solution diversity. Unlike our approach, however, Cazenille [58] used automatically defined descriptors, while we manually define and keep these descriptors fixed throughout the evolutionary process.

10.3 Graph Quality-Diversity

By integrating QD with GGP, our goal is to discover a diverse array of effective and interpretable functions. Our focus is on continuous control tasks, where the functions evolved through this process serve as control policies for tasks like maze navigation and robotic locomotion.

Our goal is to discover a set of policies which cover diverse *behaviors* (the actions taken by the policy) or diverse *structural* features (the graph which represents the policy). By doing so, we first aim to find high-performing functional graphs, leveraging the exploration of QD to better escape local optima than standard GGP. By searching for a diversity of functional graph structures, we also aim to provide a variety of solutions. As these solutions are interpretable graphs, the goal is to enhance understanding of the problem and different approaches to solving it. Finally, we aim to enhance the robustness of policy search to possible real-world scenarios such as damaged sensors.

To accomplish these goals, we propose a G-QD framework that combines GGP with QD optimization, harnessing the strengths of both approaches. In this framework, GGP serves as the foundation, offering the representation and

genetic operators, while QD functions as the evolutionary optimization method, supplying the archive for storing the population and guiding the selection of solutions for reproduction and survival.

We instantiate the general G-QD framework leveraging CGP as GGP technique and CVT-ME [434, 435], which enhances standard ME (Algorithm 2.5) with a discretization of the descriptors space based on Centroidal Voronoi Tessellation (CVT). More details about CGP and ME are available in Section 2.3.2.

10.3.1 Behavior and Graph Structural Descriptors

To enable the application of QD optimization, we define a set of descriptors to characterize each solution, i.e., each policy graph. As originally proposed for robotic control tasks [70], we employ a set of *behavior* descriptors designed to capture the aggregate of actions taken by each policy. For the maze navigation tasks, we use the final location of the agent as behavior descriptor. For robotic locomotion tasks, we use descriptors that characterize the gait of the agent based on the duration of leg contact with the ground. The use of GGP allows for the description of policy functions based solely on their graph properties as well. We therefore introduce and study *structural* descriptors that characterize the policy graphs, specifically the types and number of functions used. We fully detail the employed descriptors in Section 10.4.2.

10.3.2 Combining Multiple Descriptors

In the G-QD framework, we combine behavioral and structural descriptors to extract meaningful insights and diversify the solutions. However, since both behavioral and structural descriptors can be multidimensional, there is a potential risk of increasing the dimensionality of the QD archive to the point where uniform sampling becomes ineffective. Additionally, these two descriptor types could be treated as independent, as the same behavior can be generated by different functional graphs, and graphs with similar structures can produce different behaviors. We therefore explore two approaches for combining the two descriptor types.

Single Archive

First, we adopt a standard ME approach, using the concatenation of behavior and graph descriptors as the descriptor for defining the archive $\mathcal{A}$ (G-QD$_{\mathcal{A}}$). This results in a dimensionality growth in the archive, as the dimensionality corresponds to $n_f = n_b + n_g$, where n_b and n_g are the sizes of the behavior and graph descriptors, respectively. As CVT is used for archive discretization, the number of total cells in the archive is kept constant, even with increased dimensionality, but the hypervolume that each cell covers is therefore increased.

Multi-Archive

Secondly, we implement a multi-archive approach, similarly to [92, 58], with two independent archives: (1) the first, $\mathcal{B}$, with dimensionality n_b, for behavior descriptors, and (2) the second, $\mathcal{G}$, with dimensionality n_g, for graph structural descriptors. In this case, we slightly modify the standard ME algorithm described in Section 2.3.2 to accommodate the existence of two archives in the sampling and insertion phases. For insertion, we treat the two archives as completely independent, inserting all new individuals into both archives based on cell emptiness and local competition, as for the standard ME (Algorithm 2.5). For sampling, we consider three options: (1) sampling only from the behavior archive $\mathcal{B}$ (G-QD$_{\mathcal{B}}$), (2) sampling only from the graph structural archive $\mathcal{G}$ (G-QD$_{\mathcal{G}}$), or (3) sampling from both archives, removing duplicates if present (G-QD$_{\mathcal{B}\cup\mathcal{G}}$).

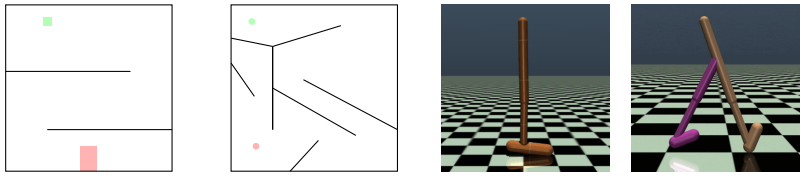
It is important to highlight that maintaining two independent archives could promote the survival of distinct individuals, as two individuals might share the same behavior descriptors but differ in graph structure, or vice versa. This setup could facilitate the development of *stepping stones* that lead to better final solutions or improve the robustness of the final individuals within the archives.

10.4 Experimental Evaluation

We conduct an experimental evaluation to assess various aspects of the proposed G-QD framework. In particular, we aim at addressing the following questions:

1. Is the G-QD framework effective at finding high-performing solutions to continuous control tasks?
2. Are the solutions found diverse? In other words, how well does G-QD explore the behavior and the graph structural space?
3. Does the framework enhance interpretability, allowing further insights into the considered problems?
4. Does G-QD increase the robustness of evolved policies and their resilience to damage?

To answer these questions, we consider four control tasks with continuous observations and actions; two are maze navigation tasks and the other two are robot locomotion tasks. We study multiple variants of G-QD to optimize graph policies on these tasks. We also consider two baseline methods for comparison, namely, a standard GA for evolving graphs and ME for the optimization of ANN policies. To understand the ability of these different methods to find useful control policies, we use the performance of the policies on the respective continuous control tasks



(a) Point maze. (b) Robot maze. (c) Hopper. (d) Walker2d.

Figure 10.1: Benchmark tasks representations.

as fitness, and we analyze the maximum fitness achieved by each method. To study the capacity of these methods to search for a wide range of diverse policies, we measure the coverage over the behavior and structural descriptor spaces. Finally, we perform an analysis on the interpretability and robustness of policies found during evolution.

We use the Kheperax [133] and Brax [120] frameworks for environment simulation, and the QDax framework [59] for the QD optimization. We release the code for reproducing all experiments together with our experimental results[1].

10.4.1 Benchmark Tasks

Navigation Tasks

The first considered navigation task is *point maze*, where a point-like agent has to navigate to the exit of a maze as fast as possible. We show the maze in Figure 10.1a: the agent is initially deployed randomly within the red area, and the task is considered solved when the agent reaches the green area. The observation is the (x, y) location of the agent $\in [-1, 1]^2$, while the action is the movement of the agent $(\Delta x, \Delta y) \in [-0.1, 0.1]^2$. The task proceeds for a total of 100 timesteps, but is terminated earlier if the agent reaches the goal. We define the fitness as the negative sum of the distance of the agent from the target location throughout the episode, to promote reaching the goal as fast as possible.

The second employed navigation task is *robot maze*, where a Khepera-like robot [284] has to move from the red point to the green point of the maze reported in Figure 10.1b. The observation $\in [-1, 1]^5$ derives from 5 sensors of two different types: (1) three limited-range lasers placed at $-\pi/4, 0, \pi/4$, which return the distance to the closest object within the range or -1 in the case where no object detected, (2) and two contact sensors covering the top-right and top-left perimeter of the robot which return 1 in case of contact with a wall and -1 otherwise. The action $\in [-0.025, 0.025]^2$ corresponds to the velocity commands sent

[1] https://github.com/giorgia-nadizar/QDax/tree/feat/cgp

to the two wheels situated to the sides of the robot. The simulation is conducted for 1000 timesteps, with early termination when the agent reaches the target location. As for the point maze, the fitness is the negative sum of the agent-target distance throughout the episode.

Locomotion Tasks

For the locomotion tasks, we use two tasks with legged agents from the Mujoco suite [422], the *hopper* and the *walker2d*, displayed in Figures 10.1c and 10.1d, respectively. The objective of the agent is to locomote as far as possible from the initial starting point, without losing its balance. The observation corresponds to various sensory inputs relative to the status of the agent ($\in \mathbb{R}^{11}$ for the hopper and $\in \mathbb{R}^{17}$ for the walker2d) and the action determines the values for the joints of the agent ($\in [-1, 1]^3$ for the hopper and $\in [-1, 1]^6$ for the walker2d). We simulate the agents for 1000 timesteps, terminating the episode earlier if the agent falls to the ground. We thus define the fitness by summing the distance run to the amount of timesteps for which the agent is stable, subtracting a small penalty term for excessive movements.

We choose these two specific locomotion tasks given the results of Chapter 9. In fact, both these tasks showed a strong tendency of GGP towards premature convergence in local optima. Thus, they constitute ideal test beds for assessing whether thanks to QD we can prevent such stagnation.

10.4.2 Experimental Settings

For each of the tasks, we study six EAs: four G-QD variants and two baseline algorithms. First, we test the four G-QD variants listed in Section 10.3.2, namely the single archive option G-QD$_{\mathcal{A}}$, and the three multi-archive approaches, differing only in terms of sampling (G-QD$_{\mathcal{B}}$, G-QD$_{\mathcal{G}}$, and G-QD$_{\mathcal{B}\cup\mathcal{G}}$). In these cases we set $n_{\text{pop}} = 100$.

Then, we consider two baselines. The first is a GA (Algorithm 2.1), with $n_{\text{pop}} = 100$, $n_{\text{off}} = 90$, $n_{\text{elite}} = 10$, and $n_{\text{tour}} = 3$, for optimizing graphs with the CGP representation. We include this baseline, which adopts the same policy representation of G-QD, to study the contribution of ME in G-QD, over the use of a standard evolutionary method that does not explicitly consider diversity. The results obtained with this algorithm should be comparable to those of Chapter 9.

The second baseline algorithm is ME for Neuroevolution (NE) with the same parameter settings as G-QD. As ME has been often used for NE, this baseline is intended as a comparison to the use of a graph-based policy. For this baseline, we consider only the behavior space $\mathcal{B}$, as there is no corresponding graph structure archive. We employ the isoline mutation variation [433] with $\sigma_1 = 0.005$ and $\sigma_2 = 0.05$; we optimize MLPs with 2 hidden layers of 64 neurons with ReLU

activation, while we use $\tanh$ as activation on the output layer. Each method is run for a total of $n_{\text{evals}} = 1\,000\,000$ evaluations and repeated 30 independent times.

For CGP, we set $n_{\text{nodes}} = 50$ and we consider the following function set: $H = \{\bullet + \bullet, \bullet - \bullet, \bullet \times \bullet, \bullet \div^* \bullet, |\bullet|, \exp \bullet, \sin \bullet, \cos \bullet, \log^* \bullet, \sqrt{\bullet}^*, \bullet < \bullet, \bullet > \bullet, \bullet < 0, \bullet > 0\}$, where $\bullet$ represents an operand, and operators marked with * are protected. The last four functions are Boolean functions, where the output is 1 if the condition is satisfied and 0 otherwise. Moreover, we add two constant inputs to each observation, namely $\{0.1, 1\}$, thus increasing all observation dimensionalities by 2. We include the constant inputs based on previous observations that CGP benefits from having constant inputs in order to combine the constants later in the graph.

For the ME archives, we use behavior descriptors of dimension n_b and graph structural descriptors (for the G-QD methods) of dimension n_g. The behavior descriptors are task dependent: we use the final (x, y) location of the agent for navigation tasks ($n_b = 2$) and the percentage of ground contact of each leg for locomotion tasks ($n_b = 1$ for hopper, $n_b = 2$ for walker2d). Conversely, the graph structural descriptors are task-agnostic, as they rely only on the graph structure. We use two descriptors ($n_g = 2$): (1) the total number of 1-arity functions in the active graph divided by n_{nodes}, and (2) the total number of 2-arity functions in the active graph divided by n_{nodes}. Each of these descriptors is defined in $[0, 1]$, but their sum cannot exceed 1. Therefore, we adjust the archives generation procedure to take this into account and avoid creating unreachable cells.

At the end of each optimization process, we perform additional analyses on the resulting policy archives. First, we evaluate the interpretability of the policies by leveraging the ϕ metrics proposed by Virgolin et al. [445] for mathematical formulae. To do so, we first extract the equations mapping the inputs to the outputs in the graph, and then compute the median value of ϕ across these equations. We then rescale the results with min-max normalization, with $\min(\phi) = 77.9$ and $\max(\phi) = 79.1$, to obtain interpretability values in $[0, 1]$: the greater, the more interpretable. We remark that, although interpretability is a highly subjective notion, in Chapter 8 we have observed good correlation between the ϕ metrics and users' preferences, thus we consider ϕ as a reasonable proxy for interpretability.

Second, we assess the robustness of solutions by disabling each sensor of the agent individually, i.e., always returning a 0 reading, and then measuring the performance of the policy with a disabled sensor. In particular, we re-evaluate all solutions (i.e., the whole final archives or the entire final population), and report the performance of the best one for each optimization condition (random seed and EA) and sensor disabled.

10.5 Results and Discussion

10.5.1 Performance

To evaluate the effectiveness of G-QD at finding well-performing solutions, we consider the fitness of the best individual in the population/archive. We display the progression of maximum fitness over evolution in Figure 10.2, where we can observe that evolution is occurring in all cases, with some differences among tasks and methods. For both navigation tasks, all optimization techniques converge to a plateau, whereas for locomotion tasks, we see that we could potentially obtain increased performance for most QD methods with additional iterations. We limit the total number of iterations based on computational cost.

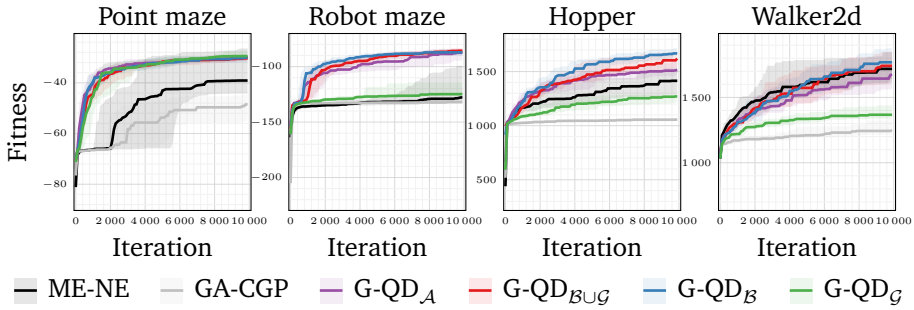


Figure 10.2: Progression of the fitness collected by the best policy in the population at each iteration; median and inter-quartile range across 30 independent runs.

To facilitate a comprehensive comparison of the methods, we present the distribution of the fitness of the best solution at the end of the evolution for each scenario in Figure 10.3. Additionally, we employ two statistical tools for further analysis: the Kruskal-Wallis H-test, which allows us to evaluate multiple methodologies simultaneously, and the Mann-Whitney U-test for pairwise comparisons. For the latter, we adjust the significance threshold using the Bonferroni correction. In all tests, we adopt equality as the null hypothesis.

In Figure 10.3, we first note the advantage of promoting diversity to achieve better solutions: all QD methods outperform the GA across all tasks. This observation is supported by statistical analysis. Secondly, we find that behavioral diversity plays a crucial role in obtaining high-performance policies. For all tasks except hopper, the fitness distributions achieved by NE, G-QD$_{\mathcal{B}}$, G-QD$_{\mathcal{B}\cup\mathcal{G}}$, and G-QD$_{\mathcal{A}}$ are statistically indistinguishable. However, in the hopper task, all distributions are statistically different, although those from G-QD$_{\mathcal{B}}$ and G-QD$_{\mathcal{B}\cup\mathcal{G}}$

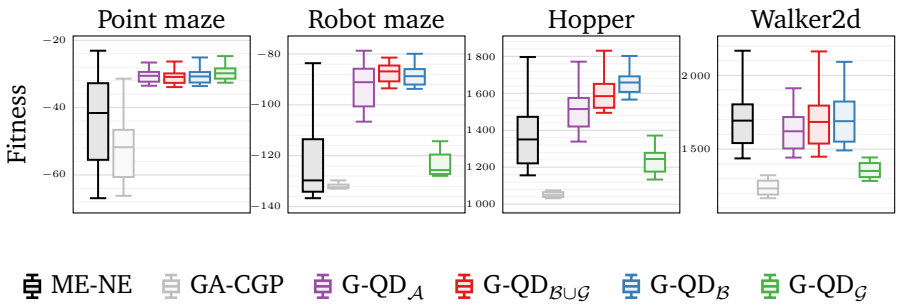


Figure 10.3: Distribution over 30 independent runs of the fitness of the best policy in the population at the end of evolution.

are quite similar. The importance of using behavior descriptors is further supported by pairwise comparisons: in all tasks except point maze, G-QD$_{\mathcal{G}}$ performs significantly worse than all other QD methods. Relying solely on graph descriptors leads to less effective results compared to when both graph and behavioral descriptors are used to search for diversity.

Summarizing, we can answer affirmatively to our first question: G-QD is indeed effective at finding high-performing solutions, provided that behavior descriptors are involved in the archive definition.

10.5.2 Diversity

In our second question, we concentrate on diversity. To evaluate this, we use the coverage of the behavior and graph structural spaces as metrics. For all multi-archive methods and the NE baseline, we calculate coverage directly from the archives where the solutions are stored. For G-QD$_{\mathcal{A}}$, we measure coverage in the $\mathcal{B}$ or $\mathcal{G}$ archives by inserting all the solutions from $\mathcal{A}$ into these respective archives. The progression of this coverage throughout the evolutionary process is presented in Figure 10.4.

The key insight from analyzing the coverage plots is that coverage is significantly impacted by both the storage method used for solutions, whether single or multi-archive, and the sampling strategy employed. Specifically, we find that sampling from a dedicated archive is essential for achieving diversity in the metric associated with that archive.

To confirm this, we can examine the coverage in $\mathcal{B}$, where G-QD$_{\mathcal{B}}$ consistently achieves the highest diversity. This method uses a dedicated archive for different behaviors and samples exclusively from it when generating offspring. Similar observations apply to the coverage in $\mathcal{G}$, where G-QD$_{\mathcal{G}}$ demonstrates superior

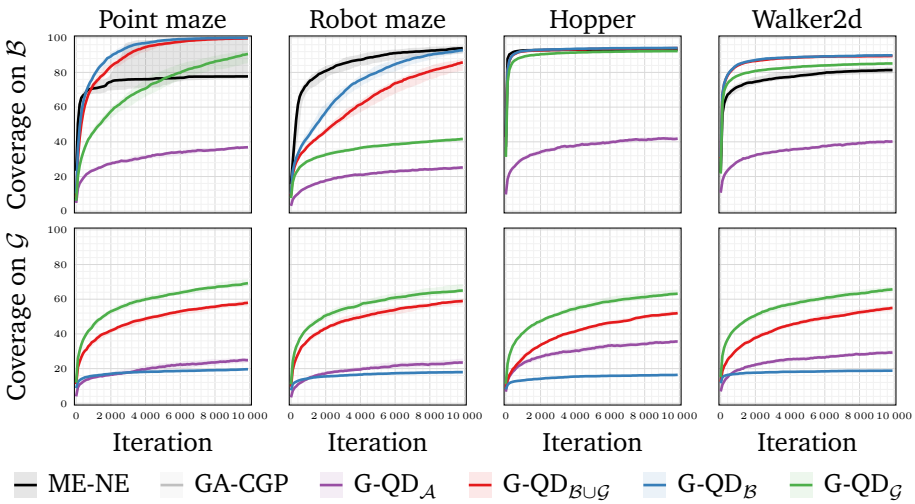


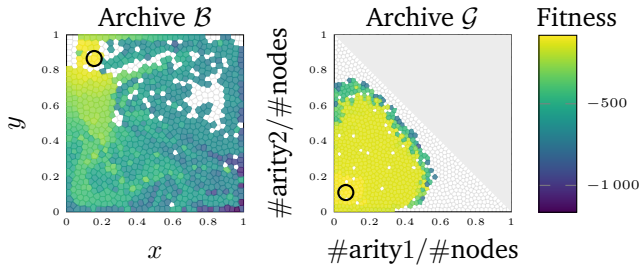
Figure 10.4: Progression of the coverage on the behavior archive $\mathcal{B}$ and on the graph structural archive $\mathcal{G}$ at each iteration; median and inter-quartile range across 30 independent runs.

diversification compared to all other G-QD approaches.

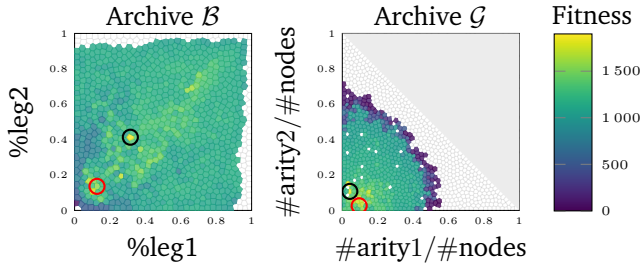
Thus, to achieve diversity in both spaces simultaneously—balancing coverage in $\mathcal{B}$ and $\mathcal{G}$—we need to consider approaches that account for both behavior and graph structural descriptors, such as G-QD$_{\mathcal{A}}$ and G-QD$_{\mathcal{B}\cup\mathcal{G}}$. However, G-QD$_{\mathcal{A}}$ presents a poorer trade-off compared to G-QD$_{\mathcal{B}\cup\mathcal{G}}$, likely due to its higher dimensionality. Focusing on G-QD$_{\mathcal{B}\cup\mathcal{G}}$, we find that it achieves coverage in $\mathcal{B}$ comparable to G-QD$_{\mathcal{B}}$ across 3 tasks, while it remains slightly behind G-QD$_{\mathcal{G}}$ in terms of $\mathcal{G}$ coverage.

In conclusion, also our second question receives a positive answer, although each G-QD approach fosters diversity in the behavior and in the graph structural space in a peculiar manner.

In addition to the quantitative analysis of coverage, we further evaluate the diversity provided by G-QD by examining two sample archives generated with G-QD$_{\mathcal{B}\cup\mathcal{G}}$, as shown in Figure 10.5. The archives presented in Figure 10.5a and Figure 10.5b reveal patterns that offer insights into both the tasks and the CGP representation. In the behavior archives, we observe a spatial relationship between descriptors and fitness: for the navigation task, the best-performing solutions are located closest to the target point, while in the locomotion task, effective gaits are clustered around the diagonal, indicating a balanced use of the agent legs. In the graph structural archives, we notice a color gradient where smaller



(a) Robot maze.



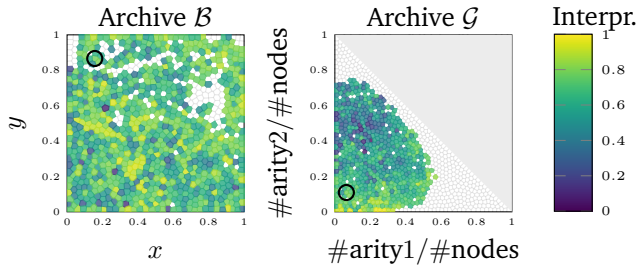
(b) Walker2d.

Figure 10.5: Sample archives from a navigation (robot maze) and a locomotion (walker2d) task, obtained at the end of optimization with G-QD$_{\mathcal{B}\cup\mathcal{G}}$. The upper-right portion of the graph structural archives is not reachable. The black circles ○ highlight the best performing policies, the red circles ○ highlight the most interpretable one. The highlighted policies are reported in Figure 10.7.

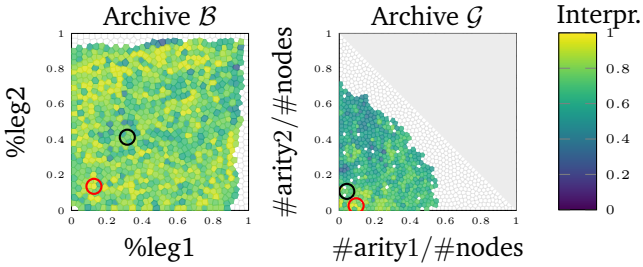
solutions tend to achieve better fitness compared to more complex ones. Although this might seem counterintuitive, it aligns with previous findings on CGP, which have shown a natural bias towards smaller solutions, irrespective of the task complexity. This pattern suggests that better solutions are more frequently found in this area of the graph structural space because it is explored more extensively. Consequently, this also explains why the coverage in the graph structural space tends to plateau at values below 100 %: extremely large solutions are rarely encountered during evolution.

10.5.3 Interpretability

Assessing whether G-QD enhances interpretability is a complex task. Although graph-based policies are inherently more interpretable (as illustrated by the poli-



(a) Robot maze.



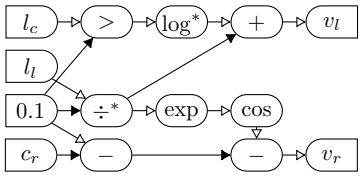
(b) Walker2d.

Figure 10.6: Interpretability of sample archives from a navigation (robot maze) and a locomotion (walker2d) task. The archives are the same shown in Figure 10.5.

cies shown in Figure 10.7), we have already discussed how interpretability is a highly subjective concept [447], and not all policies are equally understandable. Certainly, if users are presented with a broad range of high-quality and diverse solutions, they are more likely to find at least one well-performing policy that they can interpret. However, this assumes that the presented policies are diverse in terms of their interpretability.

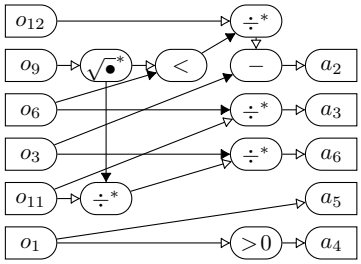
To determine if G-QD effectively diversifies in the interpretability space, we use a metric ϕ derived from a user study on mathematical formulae, which aims to capture some aspects of interpretability [445]. We conduct a qualitative analysis and visualize the archives as heatmaps of ϕ to examine its relationship with the chosen descriptors. For brevity, we present only a sample of two archives obtained with G-QD$_{\mathcal{B}\cup\mathcal{G}}$ in Figure 10.6.

Examining Figure 10.6a and Figure 10.6b, we observe a spatial dependence of ϕ on the archive $\mathcal{G}$. While the colors in $\mathcal{B}$ appear scattered, showing little pattern,



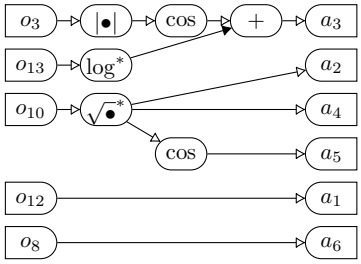
$$v_l = \frac{l_l}{0.1} + \log(l_c > 0.1)$$
$$v_r = 0.1 - c_r - \cos\left(e^{\frac{l_l}{0.1}}\right)$$

(a) Robot maze best performing policy (marked with O in Figures 10.5a and 10.6a).



$$a_1 = 0 \qquad a_4 = o_1 > 0$$
$$a_2 = \frac{o_{12}}{\sqrt{o_9} < o_6} - o_3 \qquad a_5 = o_1$$
$$a_3 = \frac{o_{11}}{o_{o_6}} \qquad a_6 = \frac{o_{11}}{\sqrt{o_9}} \frac{1}{o_3}$$

(b) Walker2d best performing policy (marked with O in Figures 10.5b and 10.6b).



$$a_1 = o_{12} \qquad a_4 = \sqrt{o_{10}}$$
$$a_2 = \sqrt{o_{10}} \qquad a_5 = \cos\sqrt{o_{10}}$$
$$a_3 = \cos|o_3| + \log o_{13} \qquad a_6 = o_8$$

(c) Walker2d most interpretable policy (marked with O in Figures 10.5b and 10.6b).

Figure 10.7: Selected policies, shown as graphs and corresponding equations, for the robot maze and walker2d tasks. In graphs, input nodes are shown with rounded angles on the right $\boxed{o_1}$ and output nodes with rounded angles on the left $\boxed{a_1}$; for inner nodes with arity 2, the second input is marked with full arrow heads $\rightarrow$; for all inner nodes, the first input is marked with empty arrow heads $\rightarrow$. For the robot maze task, the observation is $(l_l, l_c, l_r, c_l, c_r) \in \mathbb{R}^5$ and the action is $(v_l, v_r) \in \mathbb{R}^2$. For the walker2d task, the observation is $(o_1, \ldots, o_{17}) \in \mathbb{R}^{17}$ and the action is $(a_1, \ldots, a_6) \in \mathbb{R}^6$.

$\mathcal{G}$ exhibits a noticeable gradient of color. This suggests that while diversifying in the behavior space *can* lead to diversity in interpretability, it does not guarantee it. In contrast, enforcing diversification within the chosen graph structural space enables the generation of policies with varying levels of interpretability across different regions of the archive.

This corroborates the importance of considering a G-QD approach with graph structural diversity for enhancing the interpretability of the final solutions, thus yielding a positive answer to our third question.

10.5.4 Robustness

Our last analysis evaluates the solutions robustness against damage. Specifically, we assess how resilient each policy is to sensor failures. For each policy in the final archive/population, we re-evaluate its fitness n_{in} times, each time disabling the i-th input by setting it to 0. We then compute the average fitness across these evaluations and select the best individual from the archive/population based on this criterion. The results of this experiment are presented in Figure 10.8.

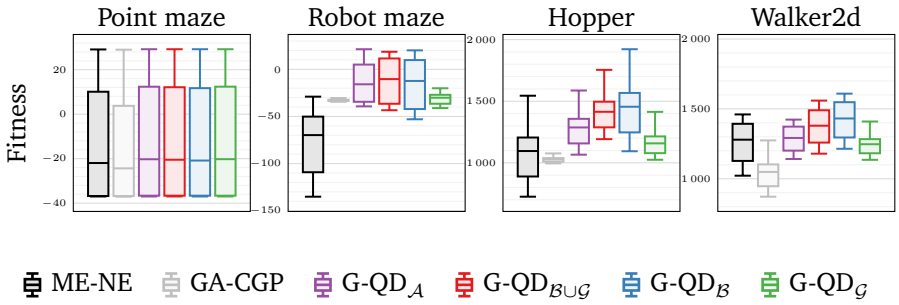


Figure 10.8: Distribution of fitness upon validation with each sensor disabled separately.

From the figure, we observe that G-QD$_{\mathcal{A}}$, G-QD$_{\mathcal{B}\cup\mathcal{G}}$, and G-QD$_{\mathcal{B}}$ all perform well in the re-assessment, demonstrating a high degree of robustness to sensor damages. In contrast, GA exhibits poor performance, as noted in Section 10.5.1, and NE also shows a significant performance decline under these conditions. This increase in robustness can be attributed to two aspects of G-QD: (1) the implicit feature selection common to all GP techniques, which likely leads to some policies partially disregarding sensory information, and (2) the diversity aspect, which, as originally proposed for QD [70], helps in finding solutions that rely on sensory information in different ways to achieve their objectives.

As in previous analyses, we also conduct a qualitative assessment of the re-

silience of various archives. In Figure 10.9, we present the graph structural archive $\mathcal{G}$ obtained with G-QD$_{\mathcal{B}\cup\mathcal{G}}$ for the robot maze task, showing both the original performance and the performance when each sensor is disabled. The plots reveal interesting patterns in how fitness changes when different sensors are disabled. Specifically, policies that heavily rely on certain sensors tend to cluster in specific areas of the archive. In these clusters, disabling the corresponding sensors results in a significant performance drop. This observation reinforces the value of promoting diversity in the graph structural space.

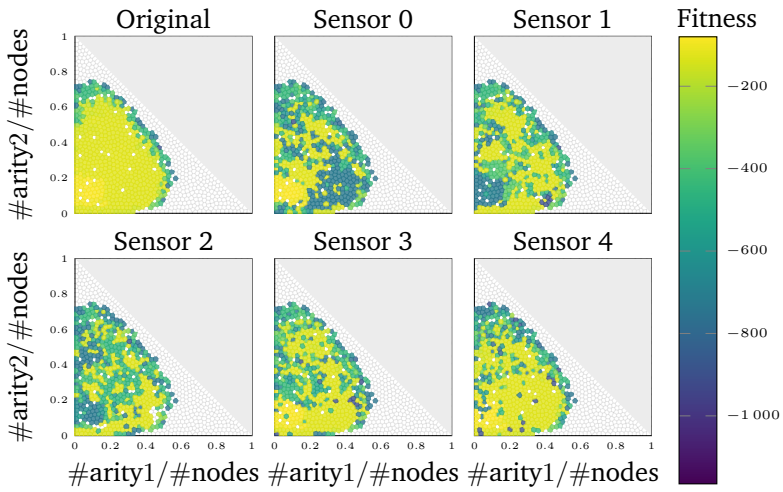


Figure 10.9: Validation of the robot maze with one sensor disabled on the graph archive $\mathcal{G}$.

10.6 Concluding Remarks

In this chapter, we introduced the G-QD framework, demonstrating the effectiveness of using QD to explore a diverse range of graph-based policies within GGP. By integrating a secondary graph structure archive with a standard behavioral archive, we showed how to uncover a variety of interpretable control policies. These graph-based policies not only surpassed ANN policies found using the same QD algorithm but also suggested that G-QD is a powerful tool for evolving high-performing and inherently interpretable policies. Additionally, we found that maintaining a diverse archive of graph policies can enhance adaptability to conditions such as sensor faults.

III

Merging Biological Inspiration with Interpretability in the Control of Embodied Agents

In this part of the thesis, we integrate the two concepts explored in Parts I and II—bio-inspiration and interpretability—into the control of embodied Artificial Intelligence (AI) agents. Our objective is to identify a paradigm of bio-inspired, interpretable embodied AI that balances the complexities and advantages of bio-mimicry with the trustworthiness of transparent control.

We begin by proposing more compact controllers for embodied agents in Chapter 11, exploring the most effective optimization approach, whether via direct optimization or imitation learning. Subsequently, in Chapter 12, we integrate graph optimization with a focus on diversity to develop embodied controllers that balance interpretability, effectiveness, and adaptability.

11

Direct vs. Imitation Learning for Interpretable Controllers

In this chapter, we compare various representations for the optimization of a controller for Modular Soft Robots (MSRs), each offering a different level of expressivity and transparency. We consider Neuroevolution (NE) for optimizing Artificial Neural Networks (ANNs), a multi-tree version of Genetic Programming (GP) for optimizing symbolic formulae, and GraphEA, an Evolutionary Algorithm (EA) that directly evolves graphs. For each representation we conduct optimization by simulating the robot to evaluate the controller fitness. Additionally, we investigate the potential of optimizing a controller via offline imitation learning, using a pre-optimized controller as an "expert". We find that, under direct optimization, multi-tree GP shows the best trade-off between interpretability and performance: while it offers a rather compact solution representation it scores competitively with NE. However, controllers obtained through offline imitation learning are consistently less effective than those evolved directly. We hypothesize that this gap in effectiveness may be explained by the possibility, given by direct evolution, of exploring during the simulations a larger portion of the observation-action space.

This chapter is based on the following publication: E. Medvet and **G. Nadizar**. GP for Continuous Control: Teacher or Learner? The Case of Simulated Modular Soft Robots. In *Genetic Programming Theory and Practice XX*, pages 203–224. Springer, 2024 [257].

11.1 Introduction and Related Works

There are numerous practical scenarios where an agent behavior is governed by a numerical dynamical system that functions as a controller: this system processes the agent observations over time and generates corresponding actions. A particularly important subset of these scenarios involves robots, whether real or simulated. In such cases, the agent observations are derived from sensory data collected from the environment, and the resulting actions are expressed as actuation values that dictate how the robot interacts with its surroundings.

It has been shown that even for complex tasks, a stateless numerical dynamical system—essentially a simple multivariate function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$—is often sufficient to enable a robot to perform the required task effectively [182, 265]. This is possible because the state necessary to solve the task, which acts as a memory of past observation-action pairs encountered by the agent, can be stored in the robot body, the environment, or both, rather than within the controller itself [315]. This is particularly true for robots with complex body dynamics, as MSRs, which can even exhibit forms of morphological computation [330].

It is not surprising, then, that ANNs, e.g., Multi-Layer Perceptrons (MLPs), have been widely utilized as multivariate functions for controlling robotic agents, owing to their strong approximation capabilities [75]. For controllers that use ANNs as multivariate functions, NE has proven to be an effective approach for optimization [116], constituting a scalable alternative to Reinforcement Learning (RL) [371].

Despite their success and large diffusion, ANNs have some limitations, particularly when interpretability is crucial. In high-risk environments, where understanding the behavior of the robot controller is critical, the opaque and over-parameterized nature of ANNs can be a disadvantage. The complexity of these networks makes it challenging to analyze or reason about the robot decision-making process [103], raising concerns in safety-critical applications.

To address this, alternative techniques that produce more interpretable controllers, while still being effective, have been explored—namely within the fields of Explainable Reinforcement Learning (XRL) and Interpretable Reinforcement Learning (IRL). Simple mathematical functions or concise graphs, as opposed to over-parameterized ANNs, can offer transparent and understandable alternatives to ANNs, particularly in scenarios where the ability to assess and explain a robot behavior is important.

One promising approach for generating such interpretable functions is tree-based GP [266, 18]. Traditionally used for solving univariate regression problems, GP has shown success in approximating complex functions, and its flexibility suggests it could be applied to multivariate functions as well [211]. In fact, numerous GP variants have been developed in which the optimized structure is

not a tree but rather a graph [117], a structure that can inherently represent a multivariate function.

There is a substantial body of literature on the use of GP variants for robot control, as discussed in Section 9.2. However, most of these works focus on relatively simple, rigid-body robots and straightforward tasks. These studies demonstrate that GP can be effective in such constrained settings where the complexity of the input-output mappings is limited, and the physical dynamics of the robot are not too intricate. We have reached similar conclusions in Chapters 9 and 10, where we explored two Graph-based Genetic Programming (GGP) techniques. While these techniques have shown some promise, our findings indicate that GP-based approaches tend to struggle when applied to more complex robotic domains, particularly those involving robots with a higher number of inputs and outputs, or tasks that require mapping intricate relationships between sensory data and actions.

Given these premises, it is unclear whether simple representations can adequately handle a complex scenario like that of Voxel-based Soft Robots (VSRs), which involve intricate body dynamics and demand precise coordination among their modules to achieve effective behaviors. Therefore, in this chapter, we aim to compare different representations—specifically, traditional ANNs versus more concise alternatives—in the context of VSRs for a locomotion task. We consider three representations: (1) a form of NE in which only the weights of the ANN are optimized through a Genetic Algorithm (GA), (2) a multi-tree version of GP, and (3) GraphEA [256], a recently proposed GGP technique.

We compare the three EAs in two ways. First, in their ability of evolving a controller through simulation, i.e., driven by a fitness function that measures the ability of the candidate controller to make the robot perform the task. Second, we evaluate their ability to evolve a controller through *offline imitation learning*, where the fitness function measures how closely a candidate controller replicates the observation-action pairs of another controller (the expert). In this scenario, the three EAs are used to solve a regression problem based on data collected by observing the expert controller during a simulation. These two scenarios differ significantly: in the first, evolution can freely explore the observation-action space during each fitness evaluation by interacting with the environment; in the second, this exploration is not possible.

The rationale for using imitation learning in this context is two-fold: (1) GP is known to be highly effective at solving regression problems, and (2) learning from an already optimized controller in an offline setting may accelerate the process, considering that the original controller might be optimized with more sample-efficient techniques (or even be available “for free”) and that regression is not a computationally expensive task. Furthermore, as discussed in Chapters 9 and 10, one of the challenges with GP techniques is their tendency to get trapped

in local optima. By imitating an optimal controller, we hypothesize that we may guide the search process toward better solutions, potentially avoiding some of these pitfalls. Although previous studies suggested that imitation learning often fails with GP [437, 150], we still consider it in this work due to the unique nature of VSRs, which differ significantly from the scenarios explored in those studies.

Our experimental results show that GP (in its multi-tree variant) performs almost on par with NE in direct evolution, while offering a much more compact and interpretable representation. However, consistent with findings in the literature, we observe that imitation learning is largely ineffective: although the resulting controllers can accurately replicate the observation-action pairs of their respective expert controllers, they fail to perform effectively when applied to the actual task in a simulated robot environment.

11.2 Evolutionary Optimization of VSR Controllers

In this chapter we rely on the distributed controller with explicit communication for VSRs, presented in Section 2.1.3. Namely, we use the non-directional communication version (i.e., the n_c communication values computed in each voxel are equally fed to all 4 adjacent voxels) and we experiment with $n_c \in \{1, 3\}$. We equip each voxel with the same controller and we set the control frequency to $f_c = 5\,\text{Hz}$ to explicitly discourage vibrating behaviors. We remark that in this chapter we consider multivariate functions as controllers, i.e., we implement the output functions $f_{i,\boldsymbol{\theta}}$ while neglecting the possible existence of a state and of a state update function within each controller.

Concerning the robot morphology, we consider VSRs with a biped shape,  , which consists of 10 voxels arranged in a 4×3 grid. As already observed in the chapters in Part I, this morphology is well-suited for the locomotion task and is generally representative enough, with minor differences observed w.r.t. other morphologies. We equip each voxel with area, touch, and x and y velocity sensors.

Given these parameters, the controller of the VSR is a function $f : \mathbb{R}^n \to \mathbb{R}^m$, with $n = 4 + 4n_c$, corresponding to the 4 sensors and the n_c communication values coming from the 4 adjacent voxels, and $m = 1 + n_c$, corresponding to the actuation value and the n_c communication values. Note that in this case the dimensionalities of the input/output spaces of the controller function, n and m, are not dependent on the VSR size, i.e., they are independent of its number of voxels.

In the following we briefly describe the considered controller representations and their optimization with Evolutionary Computation (EC). In all cases, we aim to optimize a control function that results in a VSR able to perform locomotion. To this end we drive the evolution with a fitness function q which measures the

quality of a candidate solution as a single value in $\mathbb{R}$. We detail in Section 11.3 what q exactly measures: in brief, it operates by running a simulation where the robot equipped with the controller under evaluation, i.e., the candidate f, faces a task. The larger the value of $q(f)$, the better the solution.

11.2.1 MLP-based Controller

In this variant, we use an MLP as the multivariate control function (see Section 2.2.1). We set the input layer size to n, the output layer size to m, and we use a single hidden layer with size $\lfloor 0.65n \rfloor$, as done in previous works using MLPs as distributed controllers for VSRs [338]. We use $\tanh$ as activation function for all neurons. Hence, the overall number $|\boldsymbol{\theta}|$ of parameters describing an MLP is $m(\lfloor 0.65n \rfloor + 1) + \lfloor 0.65n \rfloor (n + 1)$. In the two configurations we consider for our experiments, i.e., with $n_c = 1$ and $n_c = 3$, n and m are respectively $8, 2$ and $16, 4$, and hence $|\boldsymbol{\theta}|$ results being 57 and 214.

We optimize the MLP parameters $\boldsymbol{\theta}$, i.e., we perform a variant of NE, with the GA of Algorithm 2.1. We implement the `init()` function by sampling each individual from a uniform distribution $U([-1, 1])^p$. Concerning the variation, i.e., the `variate()` operator, we either perform crossover (with probability p_{xover}) or mutation (with probability $1 - p_{\text{xover}}$). For crossover we use an extended geometric crossover, i.e., we compute a weighted average of the two parents p_1, p_2, as $p_1 + \boldsymbol{\alpha} \odot (p_2 - p_1) + \boldsymbol{\beta}$, with $\boldsymbol{\alpha}$ sampled from a uniform distribution $U(-0.5, 1.5)^p$ and β sampled from a normal distribution $N(0, \sigma_{\text{xover}})^p$. Conversely, for the mutation we employ a standard Gaussian mutation of σ_{mut}.

In our experiments, we set $p_{\text{xover}} = 0.8$, $\sigma_{\text{xover}} = \sigma_{\text{mut}} = 0.35$, $n_{\text{pop}} = n_{\text{off}} = 100$, $n_{\text{elite}} = 0$, $n_{\text{tour}} = 5$, and $n_{\text{evals}} = 10\,000$.

11.2.2 Multi-Tree-based Controller

In this variant, we use an array of m regression trees, each representing a symbol formula, for encoding the multivariate function f. We define a regression tree for a regression problem $\mathbb{R}^n \to \mathbb{R}$ as a tree where terminal nodes are labeled with either a variable in $X = \{x_1, \ldots, x_n\}$ or a constant in $C \subseteq \mathbb{R}$ and non-terminal nodes are labeled with an arithmetic operator in H (the function set). We assume that operators in H may have different arities and that regression trees are valid in terms of arity: that is, if a node is labeled with an operator h with arity n_h, then it has exactly n_h children nodes. In this work, based on our previous experience, we set $H = \{+, -, \times, \div^*\}$, where all the operators have the same binary arity and $\div^*$ is the protected division, and $C = \{0, 0.5, 1, \ldots, 5\}$. More sophisticated options are known in literature for incorporating numerical constants in regression trees, as ephemeral random constants [339, 446]: for the sake of simplicity, we do not employ them here and leave their investigation

to future work. We denote by $T_{n,H,C}$ the set of all the possible trees defined as above.

For expressing a multivariate function $f : \mathbb{R}^n \to \mathbb{R}^m$ suitable to be the controller of a VSR, we use an array $\boldsymbol{t} = (t_1, \ldots, t_m)$ of m trees in $T_{n,H,C}$, rather then a single tree. Hence, we define f as:

$$\boldsymbol{y} = f(\boldsymbol{x}) = \left[\tanh\left(t_1\left(\boldsymbol{x}\right)\right) \ \ldots \ \tanh\left(t_1\left(\boldsymbol{x}\right)\right)\right], \tag{11.1}$$

where $t_i\left(\boldsymbol{x}\right) \in \mathbb{R}$ is the result of the application of the tree t_i to the point $\boldsymbol{x} \in \mathbb{R}^n$. We apply $\tanh$ to the result of the application of the tree to ensure that the output of f is in $[-1, 1]^m$, as required by the VSR. We denote by $T_{n,m,H,C}$ the set of all the possible arrays of m trees, i.e., $T_{n,m,H,C} = T^m_{n,H,C}$. We remark that the trees in $\boldsymbol{t}$ are independent from each other.

To optimize an array of trees, i.e., to search the space $T_{n,m,H,C}$, we propose an adaptation of Algorithm 2.1—multi-GP. As initialization we use the ramped-half-and-half method [246], repeated m times for each individual to generate an array of trees. Concerning the variation we use three genetic operators. The first, chosen with probability $p_{\text{xover}}/2$, is an element-wise standard subtree crossover, which combines the corresponding trees of the two parents. The second, chosen with probability $p_{\text{xover}}/2$, is a uniform crossover, which select each tree in the array from either parent. The last, chosen with probability $1 - p_{\text{xover}}$ is a standard subtree mutation, which is applied to all trees in the array independently. Both the initialization procedure and the genetic operators output arrays of trees in which each individual tree depth is granted to be in $[d_{\text{min}}, d_{\text{max}}]$. More details about the initialization and genetic operators are available in Section 2.3.2.

In addition, we also incorporate a mechanism for favoring diversity in the population. Namely, we simply avoid inserting in the offspring those newly generated individuals which are already part of the offspring or the parent population, i.e., those which are duplicates. According to this mechanism, inspired by [24], multi-GP re-tries to generate individuals for up to n_{attempts} times; if it fails, the duplicate individual is accepted. This diversity mechanism is particularly important since the search space $T_{n,m,H,C}$ is discrete, differently from $\mathbb{R}^p$ of the previous case, and hence the risk of premature convergence due to lack of diversity is not negligible [397].

In our experiments, and according to our previous experience, we set $p_{\text{xover}} = 0.8$, $n_{\text{pop}} = n_{\text{off}} = 100$, $n_{\text{elite}} = 0$, $n_{\text{tour}} = 5$, $n_{\text{attempts}} = 100$, $d_{\text{min}} = 3$, $d_{\text{max}} = 8$, and $n_{\text{evals}} = 10\,000$.

11

11.2.3 Graph-based Controller

In this variant, we use a graph as the multivariate function, similarly to what we have done in Chapters 9 and 10. More specifically, we consider directed

graphs with unlabeled edges whose nodes can be of four types: input, constant, hidden, or output. For a multivariate regression problem $\mathbb{R}^n \to \mathbb{R}^m$, input nodes are labeled with variables in $X = \{x_1, \ldots, x_n\}$, constants nodes are labeled with constants in $C \subseteq \mathbb{R}$, hidden nodes are labeled with arithmetic operators in H, and output nodes are labeled with variables in $Y = \{y_1, \ldots, y_m\}$. We enforce a few constraints on the structure of the graphs. Each graph g has to: (1) be acyclic; (2) contain exactly $|X| = n$ input nodes, each labeled with one different variable in X; (3) contain exactly $|C|$ constant nodes, each labeled with one different constant in C; (4) contain exactly $|Y| = m$ output nodes, each labeled with one different variable in Y; (5) have no incoming edges to input and constant nodes, no outgoing edges from and exactly one incoming edge to output nodes, and the correct number of incoming edges (corresponding to the arity of the label) to each hidden node. We denote by $G_{n,m,H,C}$ the set of all the graphs that meet these requirements.

When we compute the output $\boldsymbol{y} \in \mathbb{R}^m$ from the input $\boldsymbol{x} \in \mathbb{R}^n$ using a graph $g \in G_{n,m,H,C}$, we do as we do for regression trees. For each j-th element y_j of $\boldsymbol{y}$, we consider the tree $t \in T_{n,H,C}$ defined by the subgraph starting from the output node labeled with y_j and following only incoming edges: then, we compute the value of y_j using that t. Similarly to the case of arrays of tree, we use $\tanh$ on the output of t to make it fall in $[-1, 1]$.

In this work, based on our experience and on [256], we use the same set of constants as for the arrays of trees, i.e., we set $C = \{0, 0.5, 1, \ldots, 5\}$, and we set $H = \{+, -, \times, \div^*, \log^*\}$, where $\log^*$ is the protected logarithm with unary arity.

We search the space $G_{n,m,H,C}$ of graphs, driven by the fitness function q, by means of the GraphEA algorithm [256]. In brief, GraphEA evolves a fixed size population of graphs using a number of unary genetic operators (i.e., mutations defined over graphs) in the reproduction phase. In order to protect the innovation introduced by those operators, GraphEA employs a speciation mechanism, inspired by the one of NeuroEvolution of Augmenting Topologies (NEAT) [401]: as a side effect, this mechanism favors the diversity in the population. Moreover, GraphEA enforces structural constraints on graphs by performing the corresponding validity checks: whenever a check fails after the generation of a new individual, that individual is discarded and a new one is built. We refer the reader to [256] for further details.

In our experiments and following [401], we set $n_{\text{pop}} = n_{\text{off}} = 100$ and $n_{\text{evals}} = 10\,000$; for the other parameters, we use the default values of [256].

11.3 Experimental Evaluation

We perform a twofold experimental campaign. First, we aim at assessing the performance of the various representations proposed for “directly” evolving con-

trollers for VSRs: we describe these experiments in Section 11.3.1. Second, for gaining deeper insights in the experimental findings, we employ the same three EAs (NE, multi-GP, GraphEA) as learners from the controllers optimized through direct evolution, which act as experts: we describe these experiments in Section 11.3.2.

In both cases, we target the problem of optimizing a controller for the task of locomotion. Since we consider a 2-D scenario, locomotion is indeed directed locomotion, i.e., the VSR has to run the fastest possible along the positive x-direction. We use the average x-velocity v_x of the VSR during a simulation of $30\,\mathrm{s}$ (simulated time) as fitness, i.e., we set $q(f) = v_x$. For computing v_x, we simply consider the distance between the x-position of the VSR center of mass taken at $t = 0\,\mathrm{s}$ and its x-position taken at $t = 30\,\mathrm{s}$. Differently from most experiments conducted in Part I, here we do not discard the initial transient of the simulation for the fitness evaluation to also assess the response speed of each controller.

For increasing the difficulty of the task, we make the VSR run on a slightly uneven, rather than a flat, terrain. This way, controllers that better exploit the sensors should be more capable of adapting to the terrain unevenness, eventually being faster.

We perform the experiments using 2D-VSR-Sim [261] for the simulation. All our results are publicly available[1].

11.3.1 Direct Evolution of the Controller

For this part of the experimental campaign, we apply the three EAs described above to the problem of evolving a VSR controller for the task of locomotion. We perform 10 evolutionary runs for each EAs by varying the random seed.

Figure 11.1 reports the results of this experiment. It shows the fitness v_x of the best individual in the population during the evolution, one plot for each of the two values of n_c. Several interesting observations may be made from Figure 11.1.

First, both NE and multi-GP appear capable of evolving effective controllers within the budget of $n_{\text{evals}} = 10\,000$ fitness evaluations. This conclusion is supported by the following observations: (1) for these two EAs, the best fitness v_x curve stabilizes in the later stages of evolution, and (2) the absolute values of v_x are comparable to those reported in other studies involving similar robots and in Part I. Additionally, we confirm that they effectively produce effective gaits by visually inspecting the behaviors of several evolved VSRs[2].

Second, multi-GP appears to be on par with NE with $n_c = 3$ and slightly worse with $n_c = 1$. For the former case, multi-GP is as effective and as efficient as NE;

[1] `https://github.com/ericmedvet/2023-GPForContinuousControlAndLearning`

[2] Videos of the VSRs obtained with the first random seed and $n_c = 1$ are available at `https://github.com/ericmedvet/2023-GPForContinuousControlAndLearning/tree/main/videos`.

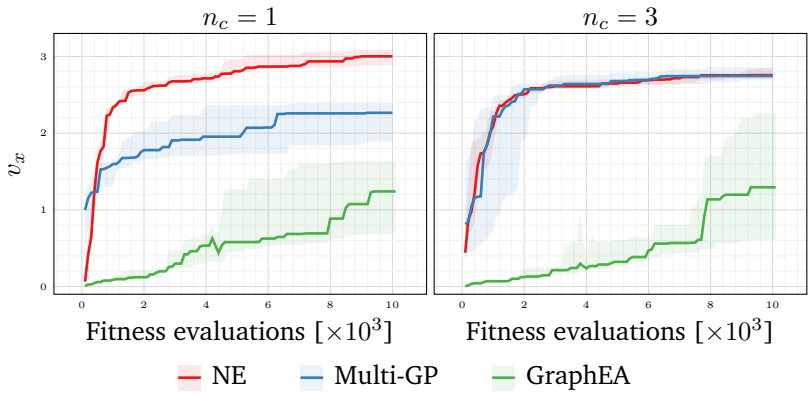


Figure 11.1: Median and inter-quartile range (across 10 runs) of the velocity v_x of the best individual along fitness evaluations.

that is, it achieves the same values of v_x in the same number of evaluations. Interestingly, with the smaller search space corresponding to $n_c = 1$, multi-GP achieves lower values of final v_x than with $n_c = 3$, while NE does the opposite. This discrepancy may be explained by the fact that a larger number of communication channels facilitates reactive behaviors through higher values of exchanged communication. These larger values are more readily achieved with multi-GP, which can utilize operators (such as $\div^*$) that easily generate large outputs from small inputs. In other words, trees might be more adept at implementing a sort of bang-bang control [29], where abrupt changes in actuation and communication values lead to effective gaits. Indeed, visual inspection of the videos of the evolved robots reveals that those controlled by trees generally exhibit less smooth movements compared to robots controlled by MLPs.

Third, unlike NE and multi-GP, GraphEA struggles to evolve effective controllers, achieving significantly lower values for v_x by the end of the evolution. Additionally, the v_x curve does not appear to converge to a plateau. We hypothesize that this lower effectiveness may be due to the default speciation mechanism settings in GraphEA, which could contribute to a slower evolution process.

Continuing our analysis, we present the size of the best solution in the population throughout the evolution in Figure 11.2. For NE, the size is represented by $|\boldsymbol{\theta}|$ for each individual. For multi-GP, the size is the sum of the sizes of the trees in the array. For GraphEA, the size is calculated as the sum of the number of nodes and edges in the graph. Figure 11.2 serves various purposes, as supporting the interpretation of the lower effectiveness of GraphEA and gaining additional insights w.r.t. interpretability.

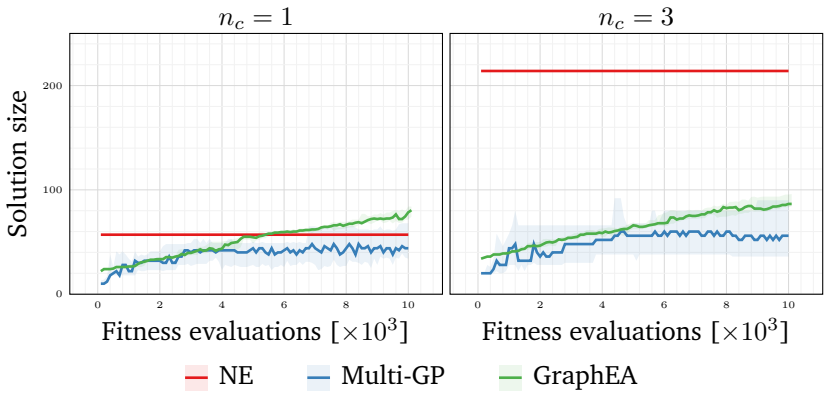


Figure 11.2: Median and inter-quartile range (across 10 runs) of the solution size of the best individual along fitness evaluations.

Two interesting observation can be made by looking at Figure 11.2. First, the size of the best arrays of trees for multi-GP seems to stabilize after about 5000 fitness evaluations. In contrast, the size of graphs in GraphEA continues to increase throughout the evolution. We interpret this continuous growth as an indication that GraphEA is significantly slower and less efficient in optimizing solutions for the case study compared to the other two methods.

Second, the size of the solutions evolved with multi-GP is only slightly larger in the $n_c = 3$ case compared to the $n_c = 1$ case. Given that the size of an array of trees is the sum of the sizes of the individual trees and that there are $m = 4$ trees for $n_c = 3$ and $m = 2$ for $n_c = 1$, it follows that the trees for $n_c = 3$ are generally simpler. This suggests that effective behavior can be achieved through simpler processing of the inputs. Moreover, this highlights how even compact controllers, potentially more interpretable, can yield effective behaviors.

11.3.2 Offline Imitation Learning

After having compared the three EAs in their ability to evolve effective controllers through direct experience of the environment where the robot is immersed, we investigate experimentally the possibility of obtaining a controller by making it "behave like" another, pre-evolved controller. For this aim, we proceed as follows:

1. we take the controllers evolved in the previous experiment and, for each of them, perform one simulation saving a dataset containing all the input-output pairs it handles;

2. for each dataset and each EAs, we perform a few evolutionary optimizations for obtaining a multivariate function that fits the dataset, driving the evolution with a fitness function measuring the error on the dataset (to be minimized);

3. we take each of the functions obtained at the previous step and put it inside a VSR, as a controller, and measure the velocity v_x it achieves in a simulation.

Concerning the first step, and considering that we actually use the f inside the controller every $0.2\,\mathrm{s}$, we obtain a dataset $(\boldsymbol{x}_i, \boldsymbol{y}_i)_i$ of $10 \cdot 30 \cdot 5 = 1500$ input-output pairs out for each controller evolved in the first experiment. We further reduce the size of each dataset by even sampling to $n_d = 375$ pairs. This way, we obtain 10 datasets for each of the three EAs, i.e., a total of $30 + 30$ datasets given by the two values for n_c.

For the second step, we use the same EAs used in the first experiment. However, we use the Average Mean Squared Error (aMSE), i.e., the Mean Squared Error (MSE) on each output variable averaged across the output variables, as fitness function:

$$q(f) = \frac{1}{m} \sum_{j=1}^{j=m} \frac{1}{n_d} \sum_{i=1}^{i=n_d} \left(f_j(\boldsymbol{x}_i) - y_{i,j}\right)^2, \tag{11.2}$$

where $f_j(\boldsymbol{x})$ is the j-th element of the vector in $\mathbb{R}^m$ obtained by applying f on $\boldsymbol{x} \in \mathbb{R}^n$. We apply each EAs to each dataset 5 times, by varying the random seed, hence performing $5 \cdot 3 \cdot 30 = 450$ runs for the case of $n_c = 1$ and 450 runs for the case of $n_c = 3$—we keep the two cases separated since the dimensionality of the dataset, and hence of the function to be optimized to fit them, is different. We call *expert* the EAs used for evolving the controller which generated each dataset and *learner* the EAs that we use to obtain a multivariate function given a dataset, i.e., to fit the dataset.

Finally, for the third step, we take each of the $450 + 450$ evolved functions and use it inside a controller of a VSR for which we measure the velocity v_x it achieves in a $30\,\mathrm{s}$ simulation.

We recall that the key difference between evolving a function through simulation and evolving one to fit a given dataset is that, in the first case, each function being evaluated is potentially able to obtain input-output pairs that facilitate the evolution. Differently, in the second case, the input-output pairs are given and cannot be changed. This difference between direct evolution and offline imitation may be of practical relevance if what the function is expected to do is to generate an effective behavior on a generally large input space, rather than "simply" approximate it on a few points.

Evolution of Functions from Data

Figure 11.3 displays the results of the first step in the form of six line plots showing how the fitness aMSE of the best solution changed during the evolution. The figure shows one plot for each combination of expert EAs and value of n_c; within the plot, the line color corresponds to the learner EAs.

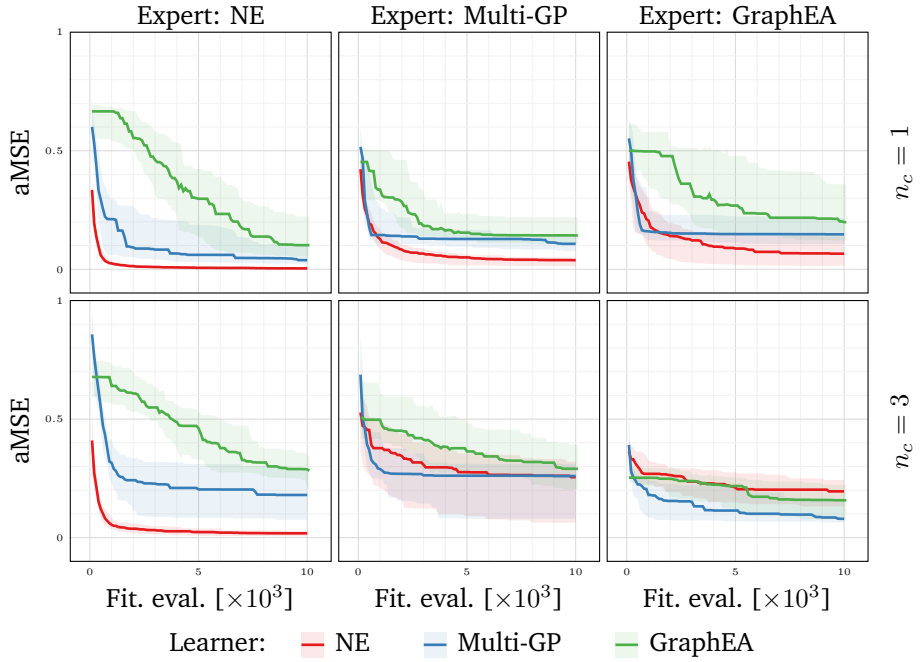


Figure 11.3: Median and inter-quartile range (across 10 runs) of the average MSE of the best individual along fitness evaluations, one column per expert, one row per n_c.

The first observation is that, generally, all three EAs appear to be effective learners, as evidenced by the steady decrease in the aMSE during evolution. In most cases, particularly for multi-GP, the budget of $n_{\text{evals}} = 10\,000$ is sufficient to reach a plateau. This is not particularly surprising, given that all three EAs have demonstrated the capability to handle regression problems effectively.

However, there are differences among the combinations of expert and learner. As a learner, NE appears to be the most effective: it achieves the lowest aMSE at the end of evolution in 4 out of 6 cases, with 3 of these cases showing particularly strong performance. NE only struggles with $n_c = 3$ when the expert is not NE.

This observation aligns with the findings discussed in Section 11.3.1: it seems that the functions evolved by multi-GP (and GraphEA) produce effective gaits in a fundamentally different manner compared to those evolved with NE. This difference is likely due to the distinct ways in which the two representations model multivariate functions.

Regarding multi-GP, Figure 11.3 illustrates that it quickly achieves a decent aMSE on the data but often reaches a plateau thereafter. Notably, the highest aMSE occurs when the expert is multi-GP with $n_c = 3$. This indicates that multi-GP as a learner struggles to effectively imitate the behaviors it produced as a expert.

Finally, consistent with the results from direct evolution, GraphEA appears to be notably less effective compared to the other two EAs. It is generally slower and achieves the highest aMSE in all cases except one.

Learned Functions Used as Controllers

We take the $450+450$ functions evolved in the previous experiment and put them inside VSRs for assessing their ability to generate behaviors that are effective for locomotion, i.e., good gaits.

We show the results of this evaluation in Table 11.1. The table shows, for each combination of expert EAs, learner EAs, and value of n_c, the 50-th (i.e., the median), 75-th, and 90-th percentile of the 50 v_x values (5 random seeds and 10 datasets) obtained through simulations. We report also the 75-th and 90-th percentile, instead of just the median, because the observed velocities were in general very low (see, for a comparison, the range of the y-axis of Figure 11.1).

The key observation from Table 11.1 is that offline imitation learning generally fails in terms of performance. None of the combinations of learner and expert achieves a median v_x value comparable to the median values obtained through direct evolution. This holds true across all learner-expert combinations. The highest median v_x is achieved when both the expert and learner are multi-GP with $n_c = 3$; however, this value is $0.15\,\mathrm{m/s}$, while it was approximately $2.75\,\mathrm{m/s}$ with direct evolution.

Examining the best controllers learned through offline imitation learning, specifically at the 90-th percentile, reveals that the most effective combination is when NE learns "from itself" with $n_c = 1$. This case of NE with $n_c = 1$ was also the top performer in direct evolution. However, a noticeable difference remains: with offline imitation learning, the 90-th percentile value is $1.79\,\mathrm{m/s}$, whereas in direct evolution, the median value is approximately $3\,\mathrm{m/s}$.

By visually inspecting the behaviors produced by controllers learned through offline imitation we find them generally quite sloppy. While the VSR is capable of generating a gait-like movement, it appears to be "out-of-sync" and clearly ineffective in advancing the robot efficiently.

	Expert→	NE			Multi-GP			GraphEA		
n_c	Learner↓	50-th	75-th	90-th	50-th	75-th	90-th	50-th	75-th	90-th
1	NE	0.12	1.03	1.79	0.04	0.17	0.54	0.03	0.16	0.56
	Multi-GP	0.00	0.02	0.07	0.04	0.14	0.24	0.02	0.06	0.11
	GraphEA	0.00	0.01	0.06	0.05	0.06	0.13	0.00	0.04	0.10
3	NE	0.04	0.17	0.70	0.12	0.25	0.48	0.01	0.08	0.23
	Multi-GP	0.00	0.00	0.01	0.15	0.20	0.36	0.02	0.20	0.35
	GraphEA	0.00	0.00	0.01	0.00	0.04	0.17	0.00	0.01	0.20

Table 11.1: x-velocity v_x obtained by the learned controllers for different experts (groups of columns) and values of n_c (groups of rows). For each combination of expert, learner, and n_c value, we show the 50-th (i.e., the median), 75-th, and 90-th percentile of the 50 v_x values obtained for that combination.

From a broader perspective, these results show that direct evolution produces functions that are much more robust than those obtained through offline imitation learning. Moreover, multi-GP appears to suffer more than NE from this lack of robustness: we attribute this weakness to the way GP represents numerical functions, which is very suitable for solving "static" Symbolic Regression (SR) problems, but may not be as effective when the functions are inserted in a dynamical system whose dynamics may be complex.

11.4 Concluding Remarks

In this chapter, we compared three EAs and their respective representations for evolving MSRs controllers for the task of locomotion. We considered a setting with controllers embedded within the voxels, able to perceive local sensory information and to communicate with adjacent voxels. As representations we considered (1) an MLP optimized with NE, (2) an ensemble of GP trees, and a (3) graph optimized with a custom EA, GraphEA. We assessed the approaches by both direct evolution, i.e., by letting the controller interact with the environment, and by imitation learning, i.e., by optimizing the controller to mimic the behavior of some other controller. Our findings revealed that a compact representation based on trees performs comparably well as more complex MLPs when trained directly; GraphEA, instead, was generally less effective. We also observed the consistent failure of imitation learning, which highlighted that, due to strong coupling between body dynamics and controller, direct interaction with the environment upon optimization is fundamental.

12

Body-Brain-Behavior Diversity for Interpretability and Robustness

On the path towards truly autonomous robots, embodied agents will require to be adaptable to unforeseen circumstances. Yet, most robotic agents still suffer from significant performance degradation when scenarios change slightly, with many even failing their tasks entirely. In contrast, organisms in nature exhibit strong adaptability, largely due to bio-diversity, which has prevented the extinction of life throughout severe environmental changes. The concept of Quality-Diversity (QD) aims to emulate this natural resilience, yielding robust results through diversification of embodied agents in the behavior space. However, in nature, diversity occurs simultaneously at multiple levels: body, brain, and behavior. In this study on simulated embodied agents—namely MSRs—we investigate these levels introducing the Body-Brain-Behavior Quality-Diversity (3B-QD) framework, to determine the most critical scope for diversity in fostering generality and robustness. We extend our evaluation to two controller representations, involving both complex ANNs and more compact interpretable graphs. Our findings for both cases confirm the paramount importance of behavior diversity, but highlight how it is closely followed by body diversity, which plays a pivotal role in specific environments. Lastly, we observe that brain diversity, while offering potential for various insights, has a minor impact on overall generality.

This chapter consists of unpublished material.

12.1 Introduction

The need for autonomous robots has become increasingly evident as they are tasked with substituting or aiding humans in complex and risky operations. In hazardous environments, robots are often deployed to minimize human exposure to danger. However, such environments are prone to unexpected situations or damages that may compromise the robot effectiveness. In these scenarios, robots must maintain their ability to function, even when faced with challenges they were not explicitly optimized for, without becoming a burden or failing in their tasks.

Observing the natural world offers a valuable lesson on adaptation and survival. Throughout evolutionary history, life has consistently evolved and adapted to survive, even amidst drastic environmental changes. A critical factor in this survival has been biodiversity: while certain species perished, others thrived because of their unique characteristics [421]. In many cases, traits that may have seemed less advantageous in stable environments became essential for survival in times of change. This is a direct result of biodiversity preserving these traits across species, ensuring the survival of life despite uncertain futures.

This biological lesson has inspired developments in the artificial domain, particularly with the emergence of techniques such as QD algorithms [69]. Algorithms like Novelty Search (NS) [219] and MAP-Elites (ME) [291] have been designed to explore search spaces by discovering diverse solutions, many of which may initially seem suboptimal. However, these diverse solutions often act as *stepping stones* for future innovation [124], eventually leading to better adaptations. In robotics, this diversity is invaluable, as a variety of solutions can help the system adapt to damage—e.g., a robot might find alternative ways to function if one of its legs becomes damaged [70].

Despite the progress made through artificial diversity, most of the research has primarily focused on behavioral diversity [293]. While this approach has shown significant benefits, biological systems diversify along multiple dimensions, including also body structure and neural connectivity [365]. These additional dimensions of diversity can also be explored in artificial systems. In particular, in modular robots, e.g., VSRs, both the controller (“brain”) and the physical body can be optimized and, in some cases, even reconfigured during their operational lifespan [475, 368].

Another crucial factor in bringing autonomous robots into everyday life is their trustworthiness [194]. Performance alone is not enough—robots must also be transparent in their decision-making processes to ensure safety and build public trust [473]. Transparency in how robots make decisions not only aids in safety but also facilitates broader societal acceptance of autonomous robots in daily use [461].

In this work, we have a two-fold goal of addressing both the possibility of achieving robustness through diversity and the interpretability problem. We consider the case-study of simulated VSRs, which are particularly well-suited as (1) they strongly mimic biological organisms with their soft compliant structure, and (2) they can be optimized both in the body and the controller. First, we explore diversity across multiple axes—body, brain, and behavior—to develop more adaptable VSRs. To this end, we propose the 3B-QD framework, illustrated in Figure 12.1, which we test for robustness by transferring robots optimized for one task to related tasks, simulating unforeseen environmental changes. Second, we involve controller representations that are inherently interpretable. Therefore, we compare traditional ANNs with a graph-based representation optimized with Cartesian Genetic Programming (CGP) [276], to assess whether transparent representations can perform on par with opaque ones.

Our results confirm the paramount importance of diversity in addressing unforeseen changes in robot tasks. Furthermore, we demonstrate that different axes of diversity contribute to generalization in unique ways, with body diversity playing a key role, followed by behavior and body diversity. Last, we highlight how with suitable optimization techniques it is possible to yield interpretable controllers matching the performance of more complex opaque ANNs.

12.2 Related Works

Bio-*diversity* plays a critical role in ensuring the robustness of life on Earth, serving as a fundamental trait for resilience and adaptability in various ecosystems [421]. This has been particularly highlighted in the context of climate change, where it is key to the survival of species under shifting conditions [289]. Drawing inspiration from biological systems, the field of robotics has also explored the concept of diversity, particularly in relation to finding better and more adaptable and robust solutions. However, the majority of the research associated with the key words "diversity" and "robotics"[1] primarily concentrates on behavioral diversity rather than structural or cerebral diversity.

Early works in this area proposed leveraging behavioral diversity to achieve improved solutions, often through NS in a multi-objective setting (with performance as the other objective). Behavioral diversity has been effectively used to address challenges such as the bootstrapping problem, i.e., the initial lack of fitness gradient, in EAs [292], as well as to avoid stagnation in highly-attractive local optima [93, 293].

Interestingly, even in cases involving modular robots like VSRs, where the design space is highly optimizable, most of the focus has remained on behavioral

[1] According to a research by keywords on Google Scholar.

diversity. For instance, Methenitis et al. [270] leveraged NS to obtain a diversity of VSR trajectories under varying environmental conditions for space exploration. Similarly, Gravina et al. [131] combined surprise and novelty to explore the behavioral space, while characterizing the diversity of the resulting morphologies post-hoc, demonstrating how morphological diversity could emerge. Lastly, Daly et al. [76] employed ME with two hand-designed behavioral descriptors to diversify VSRs, yielding innately more robust solutions.

Shifting focus to morphological diversity, several studies have examined how environmental differences drive the natural emergence of diversity [16]. Among these, the most relevant to this work is the study by Miras et al. [278], where the authors introduced descriptors for the controller, body, and behavior spaces. Yet, unlike this study, they focused solely on observing how environmental changes affect diversity across these axes, without directly fostering it. Notably, research has shown that when co-optimizing body and brain, morphological diversity tends to diminish over time [327], highlighting the need for mechanisms that specifically encourage morphological diversity during optimization.

In this direction, several approaches have been proposed to foster morphological diversity. For example, Lehman and Stanley [220] introduced the use of NS in a multi-objective setting to prevent premature convergence. Building on this, Samuelsen and Glette [374] applied NS to both the space of morphological graphs and feature spaces of modular robots, later transferring solutions to the real world and demonstrating how different morphologies were variably impacted by the reality gap [375]. In other works, diversity was incorporated directly into composite fitness functions to guide the evolutionary process [277]. Notably, De Carlo et al. [82] introduced an artificial speciation mechanism based on morphological differences to preserve diversity within evolving populations. The ME algorithm has also been effective in this domain, as shown by Veenstra et al. [436], who demonstrated its ability to promote morphological diversity with a generative encoding of 2D virtual agents. Comparative studies, such as those by Nordmoen et al. [319], further highlighted the superiority of ME over multi-objective search in maintaining diversity, as well as its usefulness in evolving robots that are more adaptable to new environments [320]. Most recently, Mertan and Cheney [269] utilized ME to generate a morphological diversity of VSRs, which were later distilled into a "general" controller capable of handling a broader range of morphologies.

Last, several works addressed multiple axes of diversity in robotic systems. These are the closest to our study. For instance, Gravina et al. [132] explored the evolution of VSRs using a "nested" ME framework, employing body descriptors for defining the ME grid and relying on variants of behavior-based NS for selection. Similarly, Zardini et al. [471] examined both brain and body features in ME for tensegrity soft modular robots, using a multi-archive approach (sim-

ilarly to our study of Chapter 10). Both these works are strongly related with ours, but both neglect one dimension which we consider: the controller in [132], and the behavior in [471]. In other studies, Medvet et al. [263] utilized hand-defined classes of behavior and morphology for VSRs, classifying individuals via an automated Machine Learning (ML) pipeline according to some extracted descriptors [338]. Their approach encompassed a speciation mechanism inspired by NEAT to maintain diversity. Relying on similar descriptors and ME, Ferigo et al. [110] investigated the interplay between evolvability, fitness, and diversity in VSRs. Our work is closely related to these, but differs from them in that (1) we focus on three axes of diversity separately (brain, body, and behavior), and (2) we consider also an interpretable type of controller for VSRs.

12.3 Body-Brain-Behavior Quality-Diversity in VSRs

In this work, we consider the body-brain co-optimization of VSRs. To this end, we rely on evolutionary optimization, both with and without an explicit focus on diversity, as we shall see in Section 12.3.1. We provide more details about fostering diversity in the evolutionary process in Section 12.3.2.

12.3.1 Evolution of VSRs

We consider the body-brain evolutionary co-optimization of VSRs with a single genome accounting for both the body and the brain of a robot. The genotype we evolve consists of the concatenation of a brain genotype and a body genotype: $\boldsymbol{g} = [\boldsymbol{g}_{\text{brain}}\ \boldsymbol{g}_{\text{body}}]$. We detail the representations of both in the following.

Brain Representations

Concerning the brain or controller part, we employ the distributed controller presented in Section 2.1.3, with all voxels having the same controller. We implement the controller with either an MLP or with a graph, in both cases realizing stateless types of controllers. When relying on the MLP, the brain genotype consists of the vector of the parameters, i.e., the weights, of the MLP, $\boldsymbol{g}_{\text{brain}} = \boldsymbol{\theta} \in \mathbb{R}^p$. Conversely, when considering a graph-based controller representation we employ CGP for its optimization. Thus, the brain genotype $\boldsymbol{g}_{\text{brain}}$ in this case consists a sequence of integers (as detailed in Section 2.3.2).

Body Representation

Regarding the body of the VSR, it consists of a polyomino enclosed in a grid of size $h \times w$. Since we conduct our experiments with the Evolution Gym (Evo-

Gym) simulation framework, each voxel of the robot can be of a different type (see Section 2.1.2). Therefore the body genotype needs to specify both which voxels are present and what is their type (i.e., material). We implement this by concatenating two parts, one encoding the former and one encoding the latter: $\boldsymbol{g}_{\text{body}} = [\boldsymbol{g}_{\text{voxels}} \ \boldsymbol{g}_{\text{material}}]$. The first part $\boldsymbol{g}_{\text{voxels}} \in \mathbb{R}^{h \times w}$ consists of real values, whereas the second part $\boldsymbol{g}_{\text{material}} \in \{1, 2, 3, 4\}^{h \times w}$ consists of integer values indicating the material of the voxel (respectively rigid passive ■, soft passive ■, horizontally active ■, vertically active ■).

The genotype-phenotype mapping is similar to that adopted in Chapter 6; it proceeds as follows. First, we reshape both $\boldsymbol{g}_{\text{voxels}}$ and $\boldsymbol{g}_{\text{material}}$ into matrices $\boldsymbol{G}_{\text{voxels}}$ and $\boldsymbol{G}_{\text{material}}$ of size $h \times w$. Second, we define a Boolean matrix $\boldsymbol{B}_{\text{voxels}}$ of size $h \times w$ with all values set to false. Third, we set to true the value of $\boldsymbol{B}_{\text{voxels}}$ corresponding to the position with the highest value in $\boldsymbol{G}_{\text{voxels}}$. Then, we set to true the element of $\boldsymbol{B}_{\text{voxels}}$ corresponding to the position with the highest value among the ones connected to the already selected ones in $\boldsymbol{G}_{\text{voxels}}$. We repeat this process until n_{voxels} are selected. Last, to obtain the VSR morphology we mask $\boldsymbol{G}_{\text{material}}$ with $\boldsymbol{B}_{\text{voxels}}$.

12.3.2 Diversity of VSRs

To foster diversity in the evolution of VSRs we propose the 3B-QD framework, which we illustrate in Figure 12.1. The framework consists in an adaptation of ME with multiple archives, each for a different type of descriptors, similar to that proposed in Chapter 10. In this case, we extract three types of descriptors from a solution, i.e., from a VSR, namely brain, body, and behavior descriptors (more details in the following). We then use them to assign the solution to a cell in the respective archive, where the solution will be inserted or not according to the standard rules of local competition of ME. Concerning the generation of new solutions, we apply representation-dependent variation operators on solutions sampled from the union (without duplicates) of the three archives. Besides the full 3B-QD framework, we also consider three variants which sample from a single archive only: Brain-QD, Body-QD, and Behavior-QD.

The rationale behind the idea of dividing the solutions into separate archives according to certain axes of diversity is the same followed in Chapter 10. Namely, two solutions, i.e., two VSRs, might differ in terms of, e.g., controller structure but might be same in terms of, e.g., morphological features: if we only maintain a single archive for one diversity dimension, certain potentially interesting solutions might be lost in the evolutionary process. This could be detrimental as lost solutions could potentially be valuable stepping stones for eventually yielding better VSRs. In addition, following the biological lesson, diversity along multiple axes could foster robustness and adaptability, thus the 3B-QD framework

constitutes a test bed for testing the hypothesis in the artificial domain.

Brain Descriptors

For the brain descriptors we differentiate among the considered controller representations.

For the *MLP controller* we consider two descriptors $(d_{\text{◐},1}, d_{\text{◐},2})$ to capture the connectivity in the ANN. More specifically, we compute the descriptors as follows. For each neuron we take all outgoing connections, take the absolute value of the synapse, and select those exceeding a threshold of 0.25, i.e., connections with weight $|w| > 0.25$. We then count the numbers of selected outgoing synapses from each neuron, and divide the value by the initial number of connections. The two descriptors $(d_{\text{◐},1}, d_{\text{◐},2})$ are the mean and the variance of this relative count across the ANN. Thus both descriptors are bounded in $[0, 1]$. Intuitively, these descriptors should capture how many "significant" connections there are in the MLP, together with how evenly they are distributed, i.e., whether some neurons are more important than others.

For the *graph controller* we use the same descriptors seen in Chapter 10. Namely, $d_{\text{◐},1}$ is the number of one arity functions in the graph divided by the available number of nodes, whereas $d_{\text{◐},2}$ is the number of two arity function in the graph divided by the available number of nodes. These two descriptors should capture both connectivity and complexity: one arity functions imply less connectivity then two arity functions, while the division by the number of available nodes indicates the relative size of the graph. Following their definition, both descriptors are bounded in $[0, 1]$, with the additional constraint that their sum cannot exceed 1.

Body Descriptors

For describing the body of a VSR we consider a metric of "activity" and one of shape. In details, we set $d_{\text{●■▲},1}$ to be the number of active voxels (i.e., the sum of horizontally and vertically active voxels) divided by the total number of voxels n_{voxels}. This descriptor is bounded in $]0, 1]$, with 0 excluded since at least one voxel needs to be active for a robot to be considered valid.

As shape body descriptor $d_{\text{●■▲},2}$ we consider the elongation of the VSR, i.e., how stretched the morphology is in its direction of maximum development. For computing the elongation we first consider the smallest ellipse that encloses the VSR morphology and then we compute the ratio of the focal distance (distance between focal points) of the ellipse over the major axis length [52]. It follows that the elongation is bounded in $[0, 1[$, with 0 corresponding to perfectly even morphologies (e.g., a circle or square).

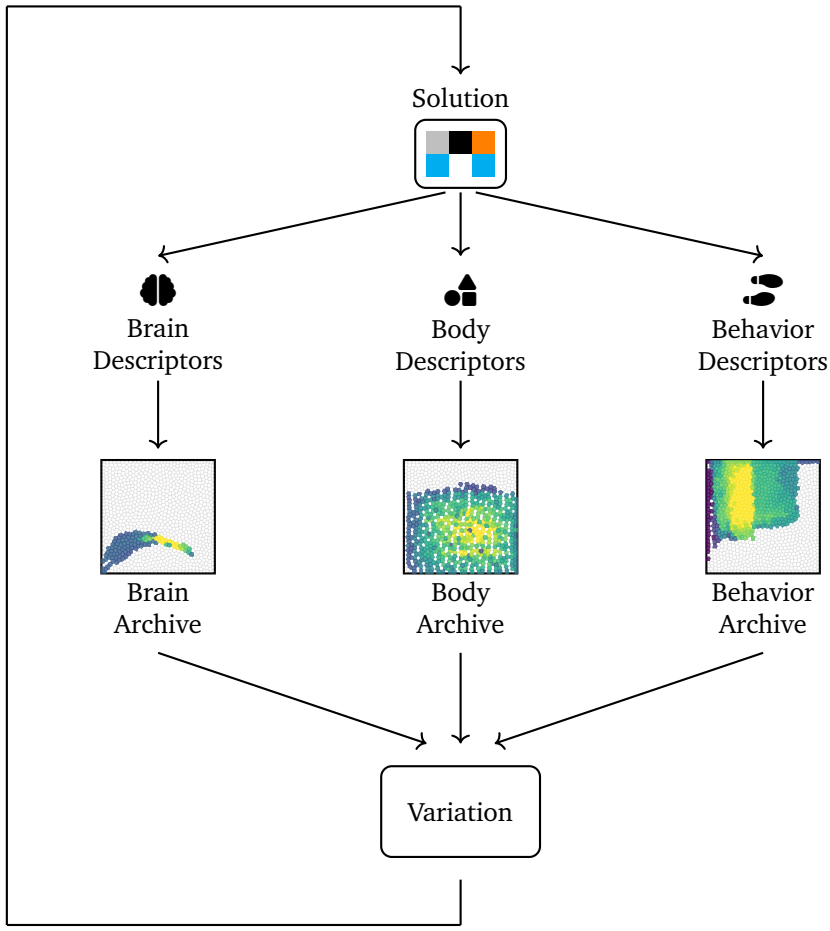


Figure 12.1: Overview of the 3B-QD framework, an adaptation of the standard ME algorithm. In 3B-QD we extract three types of descriptors from a solution—brain, body, and behavior descriptors—and assign the solution to a cell in the related archive accordingly. The solution is inserted in the archive according to the local competition rules that govern standard ME. For generating new individuals we employ variation operators on individuals selected from the union of the three archives (with duplicates removal).

Behavior Descriptors

We describe the behavior of a VSR with one descriptor related to its vertical velocity and one related to its contact with the ground, which should ideally capture the different gaits achieved by a moving VSR. For obtaining $d_{\bullet,1}$, we first compute the Fast Fourier Transform (FFT) of the signal of vertical velocity of the VSR center of mass during a simulation. Then, we compute the frequency at which the median signal energy is realized and we normalize it w.r.t. the maximum frequency. To remove the domain of too high frequencies we cap the resulting value in $[0, 0.25]$ and then rescale it in $[0, 1]$ to obtain $d_{\bullet,1}$.

Concerning the second behavior descriptor $d_{\bullet,2}$ we count the amount of simulation time steps in which the robot is in contact with the ground with at least one voxel and divide it by the total number of time steps in the simulation. Also this descriptor is bounded in $[0, 1]$ by construction.

12.4 Experimental Evaluation

We conduct a thorough experimental evaluation targeting two main questions:

1. Is the 3B-QD framework effective? In the scope of this research question, we consider the quality of the resulting solutions and the efficiency of the search process, assessing the most important axes of diversity and the coherence of results across different representations.

2. Does QD optimization within the 3B-QD framework reflect bio-diversity in terms of adaptability? Within this research question we evaluate the robustness of solutions to unforeseen changes in the environment, focusing again on the most important axes of diversity and the differences among controller representations.

The code for our experimental evaluation is based on EvoGym for the simulation and on QDax for the evolutionary optimization [59]. We make the code and the results of our experimental evaluation publicly available[2].

12.4.1 Effectiveness of the 3B-QD Framework

To test the effectiveness of the 3B-QD framework we compare all the proposed variants against a standard GA. For the comparison we consider a simple locomotion task on a flat terrain (named "Walker-v0" in EvoGym, simply "Walker" for us). We run the simulation for 500 time steps, corresponding to $10\,\mathrm{s}$ of simulated time. As already done in other works involving the distributed controller in

[2] `https://github.com/giorgia-nadizar/BBB-QD`

VSRs [267, 299], to prevent vibrating behaviors, we invoke the controller every 5 time steps, keeping the control signal fixed in between. Given the task, we consider as fitness for driving the optimization the total distance traveled by the robot along the positive x-axis Δx, computed as the difference between the final and initial x positions of the VSR center of mass.

Going more into detail about the VSRs, we equip each voxel with volume and horizontal and vertical velocity sensors (totaling 3 sensor readings). As anticipated in Section 12.3.1, we employ the distributed controller presented in Section 2.1.3, and we implement the control function with either an MLP or with a graph. In both cases the controller has $9 \cdot 3 = 27$ inputs—deriving by the sensors in the voxels belonging to the Moore neighborhood of the current voxel (a $\mathbf{0} \in \mathbb{R}^3$ is used when no voxel is present at a certain location in the neighborhood)—and a single output to control the local actuator. For the MLP we consider an architecture with 2 hidden layers of size 21 with the Rectified Linear Unit (ReLU) activation function, and a single output neuron with $\tanh$ activation. Concerning the graph-based controller we leverage CGP for its optimization, as presented in Section 2.3.2, with the following function set $H = \{\bullet + \bullet, \bullet - \bullet, \bullet \times \bullet, \bullet \div^* \bullet, |\bullet|, \exp \bullet, \sin \bullet, \cos \bullet, \log^* \bullet, \sqrt{\bullet}^*, \bullet < \bullet, \bullet > \bullet, \bullet < 0, \bullet > 0\}$, where $\bullet$ represents an operand, and operators marked with * are protected. We set $n_{\text{nodes}} = 50$ and add two constant inputs $\{0.1, 1\}$ besides the sensor readings fed as input to the graph, thus $n_{\text{in}} = 27 + 2 = 29$; we fix the output to be taken from the last node in the graph, as already done in Chapter 10.

Concerning the evolutionary optimization, we rely on the 3B-QD framework and on its three variants where sampling is only performed from a single archive (Brain-QD, Body-QD, and Behavior-QD). In addition, we also employ a GA as baseline. Thus, in total we consider 2×5 scenarios—the 2 derives from the controller representations, while the 5 comes from the amount of EAs utilized—and we repeat each optimization 10 independent times for statistical soundness.

For the GA we use the version of Algorithm 2.1, with $n_{\text{pop}} = 50$, $n_{\text{elite}} = 5$, $n_{\text{off}} = 45$, and $n_{\text{tour}} = 3$. For the 4 QD variants we set $n_{\text{pop}} = 45$ and we use the Centroidal Voronoi Tessellation (CVT) version of ME for defining the archives, with 1024 centroids identified in the allowed domain. For all EAs we set $n_{\text{evals}} = 180\,000$ to ensure fairness in the comparisons.

Concerning the genetic operators, we distinguish operators to apply on the brain and on the body part of the genotype $\boldsymbol{g}$. For the brain genome the genetic operators depend on the representation and on the EA. For the MLP controller we use Gaussian mutation with $\sigma = 0.05$ in the GA, and isoline mutation [433] with $\sigma_1 = 0.005$ and $\sigma_2 = 0.05$ in the QD variants. For the graph controller we use an int-flip mutation on the integers in the CGP genome with probability 0.1. Concerning the body genome, we leverage a Gaussian mutation with $\sigma = 0.1$ for $\boldsymbol{g}_{\text{voxels}}$ and an int-flip mutation with probability 0.05 for $\boldsymbol{g}_{\text{material}}$.

Performance Comparison

To start our comparative analysis, we report in Figure 12.2 the progression of the fitness of the best individual in the population along evolution for each EA for the two controller representations. The lines all exhibit a steadily growing trend, indicating good final performance in most cases (see [32, 267] for reference). Notably, the relationship between EAs varies depending on the representation. For MLPs, integrating QD leads to slower evolution progress and slightly worse final performance, which is consistent with the literature: QD algorithms are generally less efficient for ANNs unless facing deceptive or hard-exploration tasks [347]. In contrast, for the graph-based controller representation, while the GA starts with a quicker progression, it is overtaken by both 3B-QD and Brain-QD. It is important to mention that not all QD lines have reached full convergence, as we stop all experiments at a predefined computational budget to avoid excessive runtime.

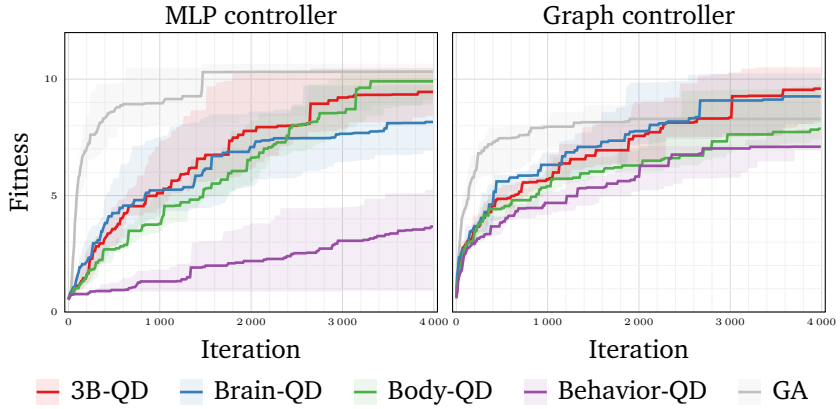


Figure 12.2: Progression of the fitness collected by the best VSR in the population at each iteration with different QD frameworks and a GA; median and interquartile range across 10 independent runs.

Moving on with our analysis, in Figure 12.3 we report the distribution of the fitness obtained by the best VSR in the population at the of evolution for all EAs and controller representations. First, we can observe interesting insights when comparing different EAs within the same representation. To back our finding we also perform the Mann-Whitney U test when comparing pairs of distributions with the equality hypothesis. For the MLP controller, all pairwise comparisons involving Behavior-QD show its statistically significant poorer performance w.r.t. other EAs (to account for multiple comparisons we adjust the statistical significance threshold to $\alpha = 0.01$), while comparisons between the other algorithms

yield no statistical significance. On the other hand, for the graph controller, from a statistical perspective all EAs perform alike, although Behavior-QD still appears quite inferior in terms of median.

When comparing the same EA across different representations, we find that Behavior-QD yield significantly better results with the graph controller, whereas neither of the other pairs show statistically significant differences. Notably, the difference obtained optimizing the MLP and the graph controller with a GA would be significant with the regular threshold of $\alpha = 0.05$, as clearly visible from the differences among the two distributions. Interestingly, the comparison between MLP+GA and Graph+3B-QD, the most successful combination for each representation, does not show significant differences. This is a key finding w.r.t. the possibility of obtaining a well-performing controller with the additional benefit of transparency.

It is also relevant to see this outcome in light of the results of Chapter 11. In this setting we observe two things: first, CGP performs clearly better than GraphEA for graph optimization for this task, and second, leveraging 3B-QD improves its performance even more.

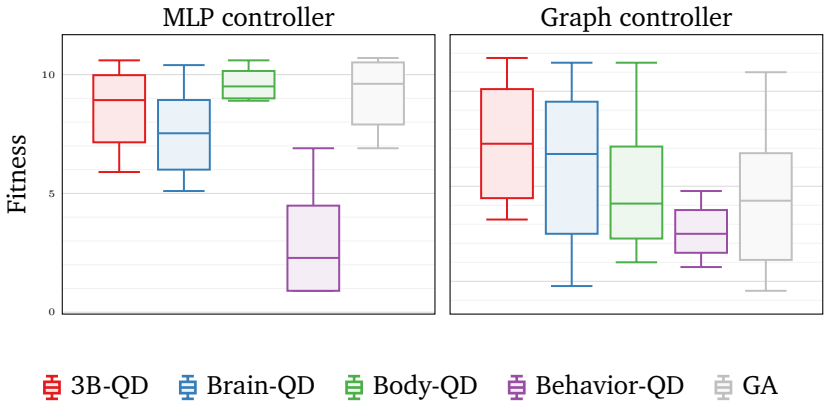


Figure 12.3: Distribution over 10 independent runs of the fitness of the best VSR in the population at the end of evolution with different QD frameworks and a GA.

To further stress the interpretability point, we report the graph policy controlling the best VSR obtained in one run of optimization with 3B-QD:

$$a_i^{(k)} = v_{x,i,\leftarrow}^{(k)} - v_{x,i,\searrow}^{(k)}. \tag{12.1}$$

The meaning of the policy is of immediate grasp: if the left neighbor of the voxel is “faster” than the bottom-right neighbor, the current voxel needs to expand,

otherwise it needs to contract. It is impressive how such a simple control policy can lead to the successful accomplishment of the task. Possibly, thanks to the rich body dynamics exhibited by the VSR, part of the computation is being performed by the robot morphology [294].

Looking more in general at the effective graph policies found by evolution, most resemble the reported one, if not in the simplicity of functionality, at least in the constrained size. Clearly, having access to a wide variety of graphs, obtained thanks to the brain diversity component, plays a key role for interpretability: when examining a successful policy, if its functioning is not clear, it is possible to consider a wide array of variants, among which it is likely to find a more transparent one.

In conclusion, the key take-home messages from these results are as follows. Overall, the 3B-QD delivers effective results in all but one case (Behavior-QD), which will be further discussed in the following sections. However, the optimization tends to progress more slowly, which is a natural consequence of the absence of strong selective pressure in QD approaches. Last, the 3B-QD framework shows the capability to optimize a transparent controller to achieve similar performance to that of an MLP. In this regard, the 3B-QD framework proves to have a positive impact on the optimization process, making a meaningful difference in reaching the desired performance.

Diversity

Another point worth noting in the analysis of the effectiveness of the 3B-QD framework is how it fosters diversity. Thus, in Figure 12.4 we report the progression of the coverage on each archive by the four QD variants. Focusing on the plots, we observe several important trends. As expected, Brain-QD and Body-QD achieve the highest coverage of their respective archives, followed closely by 3B-QD. Interestingly, for the behavior archive, 3B-QD outperforms the others in terms of coverage for the MLP controller. This result is linked to the poor performance of Behavior-QD, as previously highlighted, which also limits its ability to fully explore the behavior space (a point that will become clearer when we examine sample archives in the following). Overall, 3B-QD emerges as the best trade-off for diversifying across all spaces. Yet, it is important to note that the absolute coverage values never reach $100\,\%$. While this might seem problematic at first glance, we deliberately selected descriptors based on our specific interest in them, which explains and justifies the coverage limitations. In fact, in principle, one should choose descriptors based on the dimensions of interest, rather than aiming for full coverage of the archive.

When observing the sample archives at the end of evolution from both controller representations in Figure 12.5, several insights emerge. For the MLP brain archive, the archive shows a distinct zone with high fitness, while the rest of the

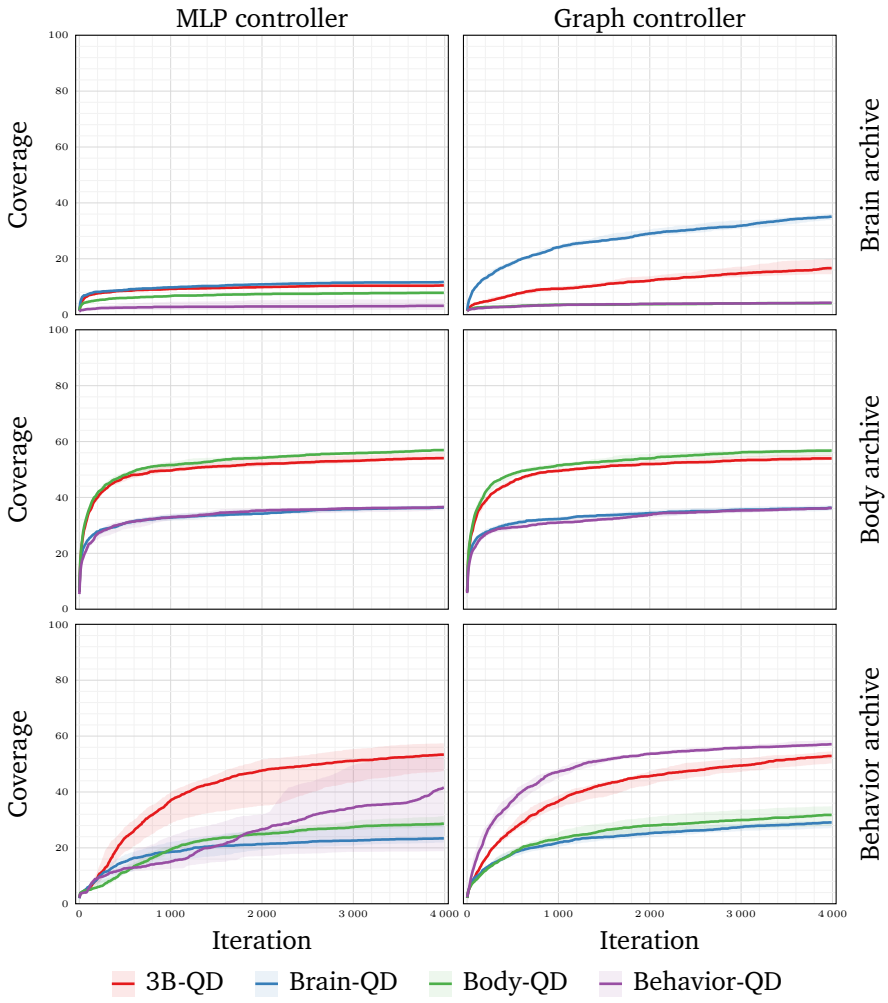
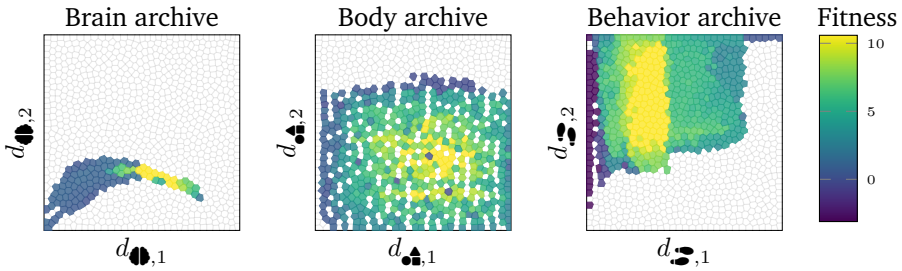
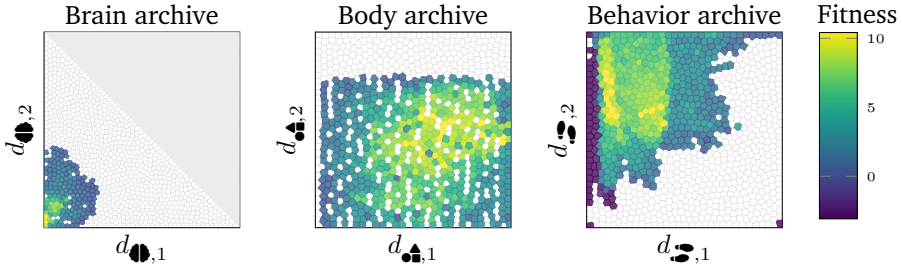


Figure 12.4: Progression of the coverage on the brain, body, and behavior archives at each iteration with different QD frameworks; median and inter-quartile range across 10 independent runs.



(a) MLP controller.



(b) Graph controller.

Figure 12.5: Sample brain, body, and behavior archives at the end of one optimization with 3B-QD with the MLP controller (Figure 12.5a) and with the graph controller (Figure 12.5b). The upper-right portion of the brain archive for the graph controller is not reachable.

covered space contains lower fitness solutions, and most of the archive is empty. This suggests that the explored space of MLPs is relatively small, with the region of MLPs capable of effectively performing the task being even smaller. Not differently, for brain graphs, due to the inherent bias of CGP towards smaller solutions, we see that most of the best solutions are clustered in the bottom left corner, while the overall coverage remains low.

Conversely, the body archives are quite similar across both representations, both in terms of coverage and color distribution. The central regions exhibit the highest fitness, while the more extreme bodies—those far from the center—tend to perform poorly under both types of controllers. Interestingly, we note that perfectly regular bodies (i.e., bodies with elongation $d_{\blacktriangle,2} = 1$) are never discovered by evolution, possibly due to a bias in the representation.

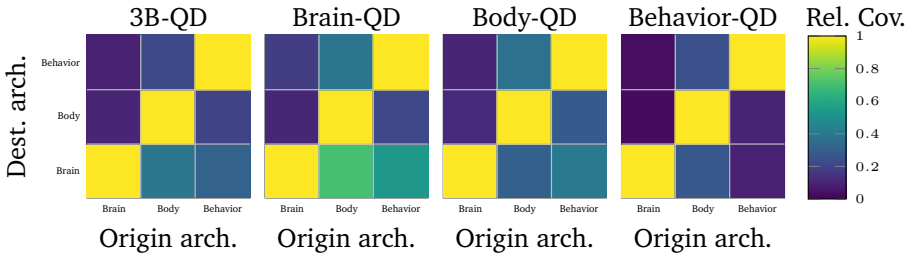
The behavior archives, although appearing similar, are not identical. Namely,

for both controllers the fittest solutions tend to be located more toward the left side of the archive, while clearly solutions with close to no contact with the ground and extremely high frequency vertical movements are not found. However, some regions are covered by one controller while remaining empty for the other. This indicates that certain behaviors are more difficult to achieve with one controller type than the other. This suggests that the considered controllers achieve good performance through different mechanisms, which explains why imitating one controller with the other is ineffective (as partially seen in Chapter 11).

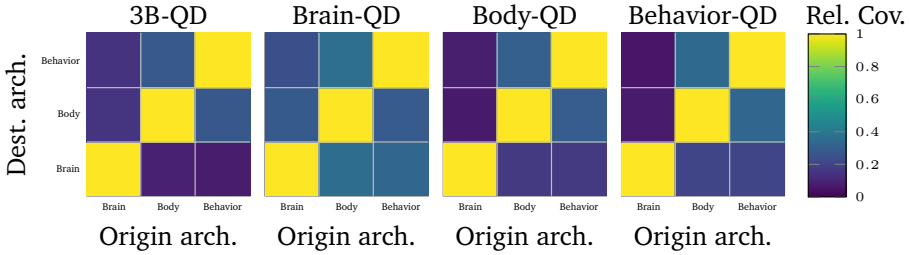
Additionally, we can make a note about the behavior archive to explain why sampling from it alone results in poorer performance. Compared to the other archives, the behavior archive retains some extremely poor solutions (since their behavior corresponds to a cell on the left of the archive), which may be sampled during evolution, clearly leading to more poor performing VSRs. When sampling from the combination of archives, these bad solutions are "diluted" within a larger set, making them less likely to be selected.

To conclude our analysis on diversity, we measure how the coverage transfers from one archive to another. The rationale behind this is to assess whether maintaining multiple archives really fosters the survival of VSRs that are diverse along a specific axis and would not be saved if that specific axis was not considered, or if in the end all archives contain approximately the same solutions. We report in Figure 12.6 the results for the sample archives shown in Figure 12.5: to compute each heat-map we take the solutions contained in an archive (reported on the x-axis), insert them in another archive (reported on the y-axis) according to the related descriptors and the standard insertion mechanism, and compute the new coverage. Last, we divide the new coverage by the initial one to take into account the maximum amount of solutions available (clearly, the coverage cannot increase). The color of cells indicates how spread one archive gets into another: a lighter color, i.e., ≈ 1, means that all individuals are in a cell alone in the destination archive, thus it is not necessary to maintain the destination archive for fostering that type of diversity; a darker cell color, i.e., ≈ 0, means that solutions tend converge into few cells, highlighting the importance of maintaining both the origin and destination archive.

Examining the figure, we can note that the majority of cells has a very dark color, confirming the importance of explicitly fostering diversity along multiple axes with the multiple archives as done by 3B-QD. The few slightly lighter cells appear for the MLP controller and the brain destination archive, which, as seen in Figures 12.4 and 12.5a we struggle to covered thoroughly.



(a) MLP controller.



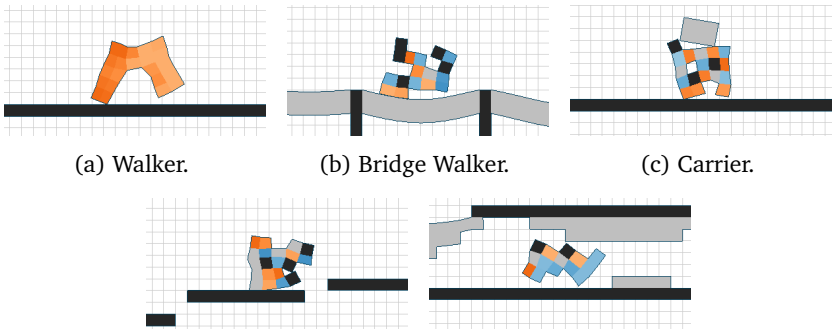
(b) Graph controller.

Figure 12.6: Relative coverage obtained by transferring the solutions contained in one archive into another archive at the of optimization for both the MLP (Figure 12.8a) and the graph controller (Figure 12.6b). The resulting coverage is divided by the original one to account for more/less full origin archives. We note that in the transfer the coverage can only decrease (multiple solutions might end in the same cell but not the other way round).

12.4.2 Adaptability of the Resulting Robots

To assess the adaptability of the evolved VSRs, we test their performance on four new tasks, which consist in modifications of the simple locomotion task they are optimized for—we report visual representations of the considered tasks in Figure 12.7. In particular, we consider (1) "Bridge Walker" in which the VSR needs to walk on a rope bridge made of several soft sections with different lengths, (2) "Carrier" in which the VSR has to carry a rectangular object while walking on a flat terrain, (3) "Platform Jumper" in which the VSR needs to jump on several flat platforms located at increasing heights along its path, and (4) "Cave Crawler" in which the VSR has to traverse caves with irregular terrain (both rigid and soft) while avoiding soft obstacles hanging from the ceiling. For the first two tasks we run simulations of 500 time steps as before, whereas for the last two tasks

we run longer simulation of 1000 time steps—in all cases we apply the control signal every 5 time steps, as done for the "Walker". The fitness in these tasks is the same as for the "Walker", i.e., the distance traveled along the positive x-axis by the VSR, for all but the "Carrier", where the fitness is given by the average between the distance traveled by the VSR and by the object, with a penalty term subtracted for dropping the object during the simulation.



(a) Walker. (b) Bridge Walker. (c) Carrier.

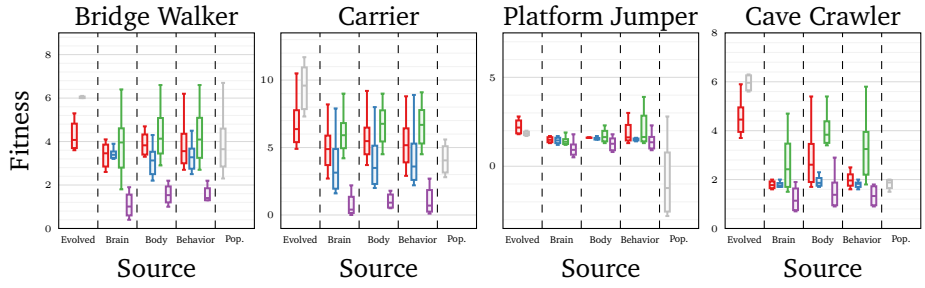
(d) Platform Jumper. (e) Cave Crawler.

Figure 12.7: Representations of the tasks considered in this work. "Walker" (in Figure 12.7a) is the original task, whereas the others (Figures 12.7b to 12.7e) are the tasks employed to test the adaptability of the evolved VSRs.

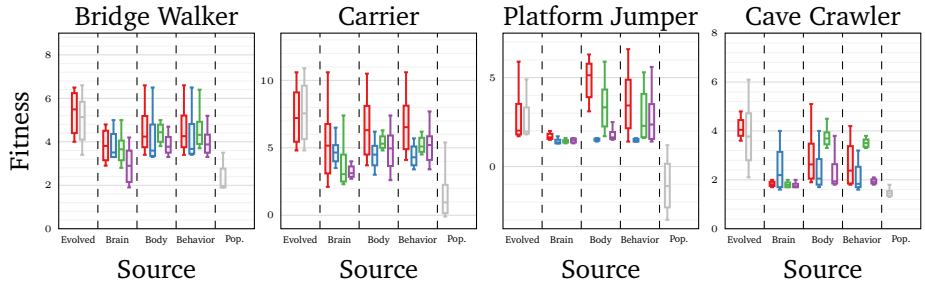
For each of these new tasks we re-assess every evolved VSR, and for each combination of archive-EA we take the best performing VSR on the new task. As baseline of comparison we also run 10 optimizations with 3B-QD and GA for each task, with the same hyper-parameters as before.

We report the results of the VSRs re-assessment, together with those from direct evolution in Figure 12.8a. The first two distributions derive from direct optimization for the new task with 3B-QD and GA, respectively. The other distributions derive from the transfer of solutions obtained with the optimization for locomotion on flat terrain. They are grouped by archive (on the x-axis, population for the GA) and EA employed for optimization (color).

Focusing on the figure, several key observations emerge about the adaptability of robots evolved through different frameworks. Most results are consistent across controller representations, showing similar trends in terms of adaptability to new tasks. One prominent finding is that body diversity plays a crucial role in fostering adaptability. The highest fitness values on new tasks are typically achieved by robots evolved using Body-QD or 3B-QD, underscoring the importance of morphological diversity. This is another point supporting the importance of morphological computation in these kind of robots [330], where the physical



(a) MLP controller.



(b) Graph controller.

EA: 3B-QD Brain-QD Body-QD Behavior-QD GA

Figure 12.8: Distribution of the fitness collected by the best individual from a given source on the 4 new tasks with both the MLP (Figure 12.8a) and graph (Figure 12.8b) controllers. Colors indicate the EA employed for optimization. The x-axis indicates if the solutions are evolved for the considered task, transferred from another archive (brain, body, or behavior) for QD frameworks, or transferred from the population obtained at the end of evolution for the GA.

structure of the body strongly influences task performance. In addition, the tasks themselves often require slightly different body configurations compared to the original simple locomotion task, such as when carrying an object or navigating tight spaces (we will see examples of VSR bodies adapted for each task in the following).

While body diversity is key, the transferability from VSRs optimized with Behavior-QD generally underperforms w.r.t. expectations. This is likely linked to the previously observed poor evolutionary performance of Behavior-QD, which hinders its ability to generalize well to new tasks. This issue is more pronounced in MLPs than in graph-based controllers. However, when the behavior archive is evolved using a more effective QD framework, e.g., 3B-QD, it still demonstrates relatively good performance in adaptability. In fact, it is intuitive to expect that behavior diversity should play a key role in adaptation: learning jumping behaviors is key for transferring to the "Platform Jumper", while slower steadier gaits appear more suited for "Carrier".

On the other hand, brain diversity appears to play a less significant role. Across all algorithms, transfers from the brain archive tend to perform worse compared to transfers from body or behavior archives, suggesting that the brain is less critical in adapting to new tasks.

The overall comparison of transfer results is in line with the outcomes observed at the end of the evolution phase. The higher the fitness at the end of evolution, the better the performance when transferring to new tasks. For MLPs, Body-QD consistently proves to be the best, while for graph-based controllers, 3B-QD yields the best results. Conversely, robots evolved using a standard GA tend to perform poorly when transferred to new tasks, especially when the tasks differ significantly from the original one. Thus, we can confirm that diversity is essential for successful adaptability.

When comparing the adaptability of evolved robots to direct optimization for new tasks, the results are mixed. While direct GA optimization tends to achieve better performance in most cases, transfers from the 3B-QD archive often perform on par with direct 3B-QD optimization, showing the value of this approach.

The take-home message from these findings is clear: diversity is fundamental for adaptability to unforeseen tasks, especially in body design. Promoting body diversity during evolution and selecting from a diverse body archive are key to achieving strong transferability. Within each QD framework, the hierarchy is evident: body archives come first in terms of importance, followed by behavior, with brain diversity being less impactful but still relevant in some cases. With the same evolutionary budget, the framework successfully tackled five tasks, matching the performance of GA on the original task and achieving satisfactory results on four new tasks. This level of performance is already sufficient for real-world applications where new tasks can emerge unpredictably, although identifying the

best VSR to deploy for the new scenario remains a practical challenge [70]. Furthermore, the evolved archives provide an excellent starting point for future optimizations, which can be used for fine-tuning and further tailoring VSRs to specific new tasks.

To complete our analysis, we report in Table 12.1 a sample of the best performing VSRs for each task, either obtained by direct optimization with 3B-QD or transferred from an archive populated with 3B-QD for the original task. We remark that within each representation all the transferred VSRs originated from the same optimization run. Upon examining the tables, we observe minimal variability within rows, indicating that different archives tend to contain slight variations of the same VSR, which still proves effective for a given task. This is especially evident for the MLP controller, where even evolved VSRs closely resemble their transferred counterparts. In contrast, VSRs with graph-based controllers exhibit more diversity within the same row. When looking across columns, for both controllers, there is more significant variability as each new task appears to necessitate distinct body morphologies to achieve optimal performance.

12.5 Concluding Remarks

In this chapter, we explored the role of diversity across multiple levels—body, brain, and behavior—in fostering generality and robustness in VSRs. To this end, we introduced the 3B-QD framework which seeks to emulate the natural resilience of organisms by diversifying evolved robotic agents across these three fundamental dimensions. In addition, we included an interpretable graph-based controller alongside a standard MLP in our experimental evaluation, to assess the generality of the findings and to test the possibility of achieving high-performing VSRs while ensuring transparency.

Our findings confirm the importance of diversity for obtaining effective and adaptable solutions. Specifically, we found that morphological diversity is the most critical factor in enhancing the generality and adaptability of agents. However, behavior diversity also plays a significant role, particularly in specific environments. In contrast, brain diversity, while it has the potential to yield interesting insights, showed a relatively minor impact on overall generality compared to the other levels. Last, our results displayed that interpretable controllers can perform on par with opaque ones provided a suitable optimization technique is employed, while granting transparency of their decision-making.

Task	Evolved	Source Archive Brain	Body	Behavior
Walker		-	-	-
Bridge Walker				
Carrier				
Platform Jumper				
Cave Crawler				

(a) MLP controller.

Task	Evolved	Source Archive Brain	Body	Behavior
Walker		-	-	-
Bridge Walker				
Carrier				
Platform Jumper				
Cave Crawler				

(b) Graph controller.

Table 12.1: Samples of the best performing VSR bodies for each task. The first column shows a sample VSR evolved specifically for the considered task. The remaining columns show VSRs evolved for the "Walker" task which performed best on the new task. Each column considers only the robots stored in a single archive, either brain, body, or behavior.

IV

Concluding Reflections

13

Concluding Discussion

The goal of this thesis was taking a stride *towards bio-inspired interpretable embodied Artificial Intelligence (AI)*. To this end, we conducted research along two main avenues, which ultimately converged together.

In Part I, we focused on taking inspiration from the biological world to improve the performance of embodied AI agents. This unfolded in exploiting the lessons learned from biology for fostering autonomy, potentially easing the porting from simulation to physical realization, and reducing over-parameterization.

In Part II, we shifted our attention to interpretability, aiming to devise (embodied) AI models providing transparency at (little to) no cost for performance. To this end we started by exploring the subjective perception of interpretability, and later leveraged these findings for realizing effective robot controllers with clearly understandable inner workings.

Finally, in Part III we merged bio-mimicry with interpretability, realizing interpretable embodied controllers for bio-inspired AI agents. To do so, we built on some of the findings of Parts I and II, combining principles of learning, evolution, diversity, and collective intelligence for fostering high-performance and adaptability in interpretable embodied AI agents.

We dedicate the next sections of this chapter to drawing the conclusions of this thesis in further detail. In Section 13.1 we retrace the research questions we laid in the introductory chapter, providing a summary of the experimental evaluation performed to answer them, and describing the insights gained. For each of the questions and corresponding analyses we also highlight the main limitations and suggest potential improvements. Finally, in Section 13.2, we consider the broader impact of this thesis and analyze possible future perspectives.

13.1 Answers to the Research Questions

13.1.1 Bio-Inspiration in Embodied AI

How can the integration of biological principles into AI for robotics enhance the performance and autonomy of embodied agents, and to what extent do in-silico findings reflect and illuminate phenomena in the biological domain?

We articulated this question into three more detailed questions, to which we provide the answers below. Anyhow, the overall answer is affirmative, in that the integration of biological principles—bio-inspired Artificial Neural Networks (ANNs), plasticity, and pruning—generally enabled improvements in the artificial realm, mostly reflecting the phenomena occurring in nature.

Research Question 1.1 *How do different models of ANNs perform as controllers for embodied agents? Namely, are more biologically plausible models more effective/efficient/general? Do they induce different behaviors?*

We answered this question in Chapters 3 and 4, by performing an experimental comparison of various ANNs for the control of Modular Soft Robots (MSRs) in several configurations, and observing notable differences across multiple quantitative and qualitative axes. In particular, we witnessed how increasing the biological resemblance of the ANNs used to control these robots impacted them positively, mitigating their fragility w.r.t. particularly difficult settings.

In Chapter 3, we systematically compared three ANN models—Multi-Layer Perceptrons (MLPs), Recurrent Neural Networks (RNNs), and Spiking Neural Networks (SNNs)—for the control of MSRs. These models differed not only in their fidelity to biological systems but also from the dynamical system point of view, such as the number of parameters and whether they maintained a state or not. We evaluated three robot morphologies combined with three sensory configurations and varying control frequencies to cover a broad spectrum of scenarios, thereby enhancing the generality of our findings. We optimized the parameters of each controller using Evolutionary Computation (EC) for a locomotion task, and we used a range of assessment metrics, including effectiveness, adaptability, induced behaviors, and search efficiency.

Our results clearly demonstrated the superiority of RNNs across all metrics, though SNNs, particularly when incorporating self-regulatory mechanisms, performed comparably in many cases. In contrast, while MLPs lagged behind, their performance was still respectable across all scenarios. This underscored the critical role of memory in the controllers performance, outweighing the significance of parameter quantity or fine-tuning. This finding was further reinforced when evaluating adaptability, where SNNs exhibited greater robustness in changing environments. The behavioral analysis provided additional insights, revealing that

the strategies different controllers used to achieve their tasks had a significant impact on their ability to generalize. For instance, some highly effective controllers displayed brittle behaviors, failing with minimal environmental changes, while slightly less efficient controllers retained most of their original capabilities even under unforeseen conditions. This chapter thus highlighted the key aspects of biological inspiration for the control of embodied AI agents—particularly memory and induced behavior. Notably, these factors often compensated for other potential issues, such as sub-optimal sensory systems, suggesting that simpler ANN models, such as MLPs, are more suitable when evaluating factors beyond controller performance. Additionally, these models could be optimized using Reinforcement Learning (RL), which could accelerate optimization through improved sample efficiency.

While our experiments spanned various configurations (morphologies, sensors, control frequencies), the generality of the results remains limited. This limitation stems from the use of a MSR simulator [261] that introduces several abstractions, most notably simulating in 2D rather than 3D. Consequently, it remains uncertain whether these findings would transfer directly to real-world MSRs. Though there have been numerous efforts to develop physical MSRs, beginning with the work of Hiller and Lipson [153] and followed by several other groups [208, 408, 218], including those exploring living matter [207], none of the existing physical MSRs can yet be finely controlled with closed-loop systems as in this study. Therefore, it is still unclear whether the behaviors observed in simulation would hold up in real-world robots, given the unpredictable dynamics of soft materials. Additionally, different ANN models may face varying challenges when implemented in physical systems. This limitation applies to other analyses as well.

To address these limitations and provide more robust results, we plan to extend our evaluation to a 3D simulator, such as VoxCAD [154], which may reduce the coarseness of the current simulations. Additionally, to account for the reality gap problem [290, 372], we could explore sim-to-sim transfer [32] or adjust physical parameters in the simulator to mimic various materials. In another study [264], we demonstrated that Neuroevolution (NE) can generally produce effective ANN-based controllers across a range of materials, suggesting that our findings could be applicable in various conditions.

Moreover, the conclusions reached in this and other chapters have broader implications beyond robotics, particularly for virtual agents in different domains. For example, the importance of an internal state for ANNs to function effectively with limited sensory input is a principle that extends across fields. While specific results from simulations may not directly translate to real robots, the overarching ideas and methodologies are applicable across a variety of domains.

Chapter 4 was conceived as a natural extension and deeper analysis follow-

ing the comprehensive investigation in Chapter 3. In this chapter, we transitioned from a centralized controller to a distributed, truly embodied controller inspired by the concept of Neural Cellular Automata (NCA). This framework closely aligned with the paradigm of collective intelligence, as each module in the system was capable of autonomous decision-making. We tested this architecture using two different ANN models: MLPs and SNNs. We excluded RNNs from this study for two reasons: (1) to reduce the computational load of the experiments, and (2) because the distributed controller inherently introduced a form of recurrence through inter-module communication. As in the previous chapter, we experimented with three morphologies but limited the analysis to a single sensory configuration. Our experimental setup mirrored that of the previous chapter, although it was less exhaustive.

Our findings confirmed the advantages of bio-inspired ANNs, demonstrating that the internal state of the ANN was more significant than the recurrence induced by communication among the voxels. While both MLPs and SNNs leveraged recurrence, SNNs consistently outperformed in terms of adaptability and effectiveness. This result also reflected the behavioral differences between controllers, much like what we observed in the previous chapter.

The limitations of this study are similar to those in the earlier chapter. However, from a feasibility standpoint, this chapter presents some advancements over Chapter 3. Notably, it eliminates the need for a central control unit, reducing the complexity of wiring or information transfer between modules of MSRs. Additionally, by designing all modules with identical capabilities and equipping each with a simple local controller—possibly implemented via the newly proposed stretchable Arduinos [460]—the fabrication process could become simpler, promoting scalability. This approach could also potentially lead to the automatic assembly of modules into robots, either via external devices [314] or through self-assembly [368]. Moreover, if local computing hardware permitted, these modules could be equipped with multiple controllers and select the most appropriate one based on their assembly configuration. We explored this possibility in a separate study [300], showing how it could enhance damage response, allowing the system to remain functional even if individual voxels were detached or lost during operation.

Research Question 1.2 *Can we replicate the biological phenomenon of plasticity in embodied agents? Do embodied agents benefit from plasticity? In what terms do they differ from non plastic ones?*

We tackled this question in Chapters 5 and 6, by simulating brain plasticity, in the form of learning, and morphological plasticity, in the form of development, in MSRs, respectively. We observed significant and constant benefits arising from brain plasticity, which endowed MSRs with the capability of achieving special-

ization to enhance their performance. Conversely, we noted more sporadic improvements constrained to specific scenarios for development, where we found the effectiveness of the MSRs to be strongly bounded to the development schedule applied.

In Chapter 5, we introduced brain plasticity into the MSRs controller through the use of Hebbian learning. We employed an embodied form of controller, applying the same set of rules across all modules, leveraging the benefits discussed in previous sections. We tested this controller paradigm within the context of body-brain co-optimization of MSRs, once again using EC to optimize locomotion performance. Beyond evaluating the MSRs effectiveness, we also assessed the occurrence of specialization within the modules.

Our findings showed significant performance improvements thanks to the introduction of brain plasticity, which also accelerated the optimization process compared to non-plastic controllers. In addition to the performance gains, we observed a specialization phenomenon, akin to totipotency. Although all modules started identical, their experiences—mainly shaped by their location within the robot—caused them to differentiate and specialize for their specific roles.

Given the similarities in experimental settings, this chapter shares the same limitations as the previous studies. One specific challenge is whether Hebbian rules could be applied with sufficient precision in physical robots to make the weight updates effective. However, Qiu et al. [350] suggest that neural plasticity, including Hebbian learning, could reduce the sensitivity of controllers to the reality gap. As a result, we view the incorporation of brain plasticity as a step forward in making physical implementations of MSRs more feasible. Moreover, the specialization observed in the modules could further enhance adaptability by allowing robots to adjust to damage without requiring explicit resets. For example, in case of an actuator damage, a broken module could be relocated to a less critical area of the robot, while a functioning module could take its place: the controllers would then re-specialize based on new stimuli and learning rules. From a fabrication perspective, these findings could simplify the process as well. All modules could be manufactured identically, with the same Hebbian rules, allowing their roles to be shaped by their experiences rather than predefined functions. This possibility could enhance scalability in manufacturing.

In Chapter 6, we explored morphological plasticity by simulating the process of development in MSRs. To achieve this, we encoded various development functions and integrated them with both embodied closed-loop and open-loop controllers. We optimized these functions using EC for locomotion, with a focus on identifying the optimal scheduling for the development of MSRs, examining whether it mirrors the patterns seen in nature. In addition to measuring cumulative performance, we analyzed how the performance evolved across different stages of development.

Our findings revealed a striking similarity to biological growth patterns. The best-performing agents were those that developed (i.e., grew) earlier and then maintained a stable body for the remainder of the simulation. This pattern was achieved despite a trade-off in actuation power: larger, more powerful MSRs did not outperform smaller ones that had followed a more efficient development schedule. These observations are particularly notable for their implications in replicating real-world phenomena with MSRs.

In summary, this chapter demonstrated how real-world biological processes can be replicated in-silico, showing that MSRs can resemble living organisms from multiple perspectives. However, it is unlikely that these findings can be directly transferred to physical robots. A major limitation of this study is that physical MSRs lack the capability for autonomous module growth. As a result, external intervention would be required at each stage of development, severely limiting the autonomy of the robots. Nevertheless, we highlight the potential of applying these findings in reverse: if a MSR had undergone development and successfully completed tasks with fewer modules, it could use that experience to continue functioning in the event of damage or module loss, without needing any modifications to the controller. This presents another potential strategy for addressing unforeseen damages in MSRs.

Last, given the promising results from investigating brain and body plasticity separately in Chapters 5 and 6, a particularly intriguing future direction would be to combine these two aspects into a unified framework. This remains an open area for further research.

Research Question 1.3 *Is it possible to leverage the biological phenomenon of synaptic pruning to reduce over-parameterization in the ANNs used to control embodied agents? Does pruning hinder the performance or does it promote adaptability to unforeseen circumstances?*

We answered affirmatively to this question in Chapter 7, by implementing the pruning mechanism in the controllers of MSRs, observing how, even with a significant portion of synapses removed, the robots would retain most of their abilities, even w.r.t. adaptability.

In this chapter, we considered various controller configurations (both centralized and embodied) and robot morphologies, and applied pruning with different methods and pruning ratios. We optimized MSRs for locomotion using EC, with pruning applied during the optimization process to ensure that the controllers adapted to the removal of a substantial portion of their synapses. We evaluated the performance of the optimized MSRs in terms of effectiveness, robustness to environmental changes, and the emergence of distinct behaviors.

Our findings showed that even after removing a large fraction of synapses—up to $75\,\%$—the controllers could maintain full functionality, as long as prun-

ing was applied according to a reasonable scheme throughout the optimization process. This demonstrates that ANNs can adapt to operating with far fewer parameters than expected, even in the context of controlling embodied agents. This mirrors the behavior of biological organisms, whose brains naturally undergo synaptic pruning after an initial phase of excessive synapse formation. Interestingly, we observed that certain ANNs could function effectively both with and without pruning if optimized accordingly. In these cases, pruning typically resulted in only minor performance degradation, although we noted clear behavioral differences between the pruned and non-pruned phases. This finding is significant for robustness: a controller that is deployed with full synaptic connectivity but experiences wiring damage could still retain most of its functionality in its pruned state.

In terms of limitations, this study shares many of the challenges noted before. However, pruning could offer a practical way to reduce wiring and connections in the physical implementation of MSRs, which could lower production costs and increase speed when suitable hardware is employed. Additionally, pruning could highlight the flow of information within the ANN, making it easier to identify critical components, such as key sensors, that might require extra attention during fabrication. This could also be a preliminary step towards interpretability, in that it could help to better understanding the decision-making processes in MSRs.

13.1.2 Interpretability in Embodied AI

How can we achieve interpretability in the control of embodied agents without sacrificing performance and considering the subjective nature of this notion?

We expanded this question into two separate questions, which we answer below. Overall, our findings indicate that it is feasible to introduce interpretability into the control of embodied agents without significantly compromising performance. This is largely due to the use of symbolic formulae or graphs, which are generally regarded as interpretable, regardless of subjective factors.

Research Question 2.1 *Can we exploit a human-in-the-loop system to discover AI models that are both well-performing and interpretable for a given user? What elements are needed for such a system to be most effective?*

We answered positively to this question in Chapter 8 building on an existing framework which combines optimization to find Machine Learning (ML) models that fit given data while learning an interpretability estimator for a given user. We found various elements playing important roles in the approach, highlighting that the involvement and interest of users is among the main factors.

More specifically, we focused on Symbolic Regression (SR), where the goal is to discover a mathematical formula that best fits a given dataset. We framed this

as a bi-objective problem, with the second objective being to maximize the interpretability of the formulae according to individual user preferences. To achieve this, we utilized an interpretability estimator (implemented with an ANN) and integrated Active Learning (AL): during the model search process, users were asked to rate formulae based on their perceived interpretability. We conducted a series of experiments simulating user preferences according to predefined criteria to evaluate the most effective ways to encode ML models, initialize the interpretability estimator, and select which formulae to present to the user for feedback. Additionally, we assessed the robustness of the framework under different user engagement profiles and tested it with real users.

Our results in simulation revealed a trade-off in encoding complexity: more granular encoding required significantly more user feedback to train the estimator effectively, increasing the effort needed from users. Furthermore, we observed variability in the interpretability-performance trade-offs depending on individual user preferences and engagement levels, underscoring the importance of personalizing interpretability estimates for different users. However, when tested with real users, our personalized framework did not yield noticeable benefits. Instead, user preferences aligned more closely with fixed estimators, such as formula size, likely due to the limited personal interest in the experiment.

This outcome highlights a key limitation of our study: although results in simulation were promising, the real-world experiments did not align with those findings. We attribute this discrepancy to two factors: (1) the users had no incentive to engage deeply with the task, and (2) they were unfamiliar with the problems and the meaning of the variables. To address this issue, we envision to deploy the framework in real-world scenarios where users have a strong interest in data fitting for practical applications, such as in the medical field or for scientific discovery. In these contexts, considering the units of the variables in the formulae would also be crucial, and should be integrated into the model search algorithm employed [358].

Another point worth noting is the strong correlation between real user results and fixed notions of interpretability, such as formula size. This suggests that the complex framework we developed is best suited for more intricate scenarios where specific interpretability criteria are critical (e.g., in the medical domain). Conversely, in simpler use cases, relying on widely accepted notions of interpretability may suffice.

Research Question 2.2 *Can we leverage graph optimization techniques to find graph control policies that are both interpretable and effective? Do we need to explicitly promote interpretability or can it emerge naturally? Is diversity a suitable avenue for this goal?*

We addressed this question in Chapters 9 and 10, where we relied on Graph-

based Genetic Programming (GGP), a graph optimization technique, to find effective yet interpretable graph policies. We found interpretable policies without explicit promotion of it thanks to the chosen optimization technique, and we improved both effectiveness and interpretability leveraging Quality-Diversity (QD) algorithms.

In Chapter 9, we began by applying two GGP techniques to derive graph-based policies for various continuous control benchmark tasks involving rigid robots with differing input/output spaces dimensions. We also conducted a bi-objective optimization, where the second objective—graph size—was introduced to explicitly promote interpretability. In fact, as demonstrated in the previous chapter, for simpler cases, a coarse interpretability metric, such as formula size, can often suffice.

Our findings revealed only minor performance inferiority for graph-based policies on simpler tasks and some more complex problems, compared to opaque ANN-based policies optimized using state-of-the-art RL algorithms. Furthermore, the resulting graphs were naturally constrained in size due to the optimization techniques employed. In fact, incorporating graph size as a secondary objective hampered the search process, leading to the discovery of simple but ineffective graphs, as the size objective was easier to pursue. To address this, future work should explore more advanced bi-objective search algorithms to avoid such pitfalls.

The outcomes of this study should serve as a starting point to raise awareness within the robotics community that high performance does not always need to come at the expense of interpretability. However, these results are limited to simulation, and it remains unclear whether transparent controllers would transfer as effectively as ANNs to real-world settings. Nevertheless, we generally expect ANNs to be more fragile due to their reliance on extensive parameter fine-tuning. Future work should test this hypothesis on physical robots, as many of the considered simulated scenarios have real-world counterparts. In particular, it would be interesting to evaluate which types of policies are more or less affected by the reality gap phenomenon [372], potentially even introducing transferability as an explicit objective in the search process [196, 197].

Another limitation of this study is the poor sample efficiency of evolutionary optimization compared to RL, which made GGP search more costly. However, it is important to note that the RL algorithms we used exploit dense rewards, whereas GGP only considers the cumulative episode reward. A potential solution to mitigate the sample inefficiency of GGP would be to start the evolutionary search with shorter simulations, i.e., episodes, gradually extending them as evolution progresses and the policies become more refined.

In Chapter 10, we built upon the findings of Chapter 9, combining GGP with QD to solve two navigation and two locomotion tasks. We leveraged QD to di-

versify policies based on two key aspects: the behaviors they induce when controlling agents and the structural features of their graphs. The rationale behind this approach was to discover stepping stones that could guide the evolutionary process toward high-performing solutions, while also finding policies that meet different needs.

Our results demonstrated that behavioral diversity played a crucial role in enhancing the performance of solutions, aligning with the conclusions of much of the QD literature. Additionally, graph diversity improved interpretability, allowing users to select the most insightful graphs, offering a better understanding of the problem and highlighting key decision-making mechanisms. Unsurprisingly, the best outcomes were achieved by balancing these two aspects of diversity, fostering both performance and interpretability simultaneously. Moreover, as a side effect of combining the natural feature selection in graph policies with QD, we observed increased robustness to sensory damage, despite not specifically optimizing for it.

In summary, this study further supports the idea that interpretability and performance are not mutually exclusive, with diversity playing a pivotal role in promoting their coexistence. In addition, having a diverse set of policies may aid in generalizing to real-world scenarios more effectively than relying on a single policy [70]. Yet, identifying which policies would perform best in real-world settings might not be trivial and might require testing all of them on physical robots. Still, we expect that some behaviors would transfer better than others, and it might be possible to characterize this with only a few real-world trials.

Lastly, we remark that our results are strongly dependent on the descriptors chosen to characterize behaviors and graphs. In fact, selecting meaningful and representative descriptors is generally challenging—some studies even proposed to identify them automatically from data gathered during simulations [68]. While an automated approach might be a viable strategy for characterizing behaviors, it would likely be less effective for graph structures. For graphs, descriptors should distinguish solutions based on features that make them more or less appealing to a given user, making user-driven selection more appropriate.

13.1.3 Integrating Bio-Inspiration and Interpretability in Embodied AI

Can we identify a paradigm of bio-inspired interpretable embodied AI that balances the complexity and the benefits of bio-mimicry with the need for transparency for trustworthiness in control?

We broke this question down into two more specific inquiries, addressed in the following. In general, our findings suggest a positive answer, as we successfully combined bio-inspired design with interpretability in the control of MSRs,

yielding significant results.

Research Question 3.1 *Is it feasible to rely on more compact controllers for MSRs? What approach is the most effective for their optimization between direct optimization and imitation learning?*

We answered affirmatively to this question in Chapter 11, relying on symbolic formulae and graphs as MSRs controllers. We found direct optimization to be the most effective strategy for finding effective controllers, observing a general failure of imitation learning for the task.

In this chapter, we explored an embodied controller design with varying degrees of inter-module communication, applied to a fixed robot morphology. We compared three controller implementations—ANNs, symbolic formulae, and graphs—all optimized using EC for a locomotion task. We initially applied direct optimization and later experimented with offline imitation learning, using data collected from other controllers experiences. We assessed the effectiveness of the resulting controllers, with special attention to their size.

Our results revealed that, with direct optimization, symbolic formulae performed almost on par with ANNs, while maintaining a significantly smaller size. On the other hand, the considered graph optimization technique generally failed to produce effective controllers. In terms of imitation learning, although all representations were able to achieve low error relative to their "teachers", they all performed poorly when deployed as controllers. This underlined the importance of direct interaction with the environment for optimization, as it allows for exploration across a broader range of scenarios.

This final outcome is partially negative. We had anticipated that imitation learning would offer a viable way to reduce optimization costs, given that regression from an expert is computationally cheaper than extended interactions with the environment. The idea was that an expert could be trained using more cost-effective and sample-efficient methods, such as RL, and then that knowledge could be transferred to a controller with a different representation. Yet, this approach did not yield the expected results. Anyway, even though the controllers learned through imitation were not highly effective, they could still serve as a foundation for bootstrapping the optimization process, potentially making it faster and more efficient. This remains an interesting avenue for future research.

Concerning the reality gap problem, we anticipate that MSRs with more compact controllers could be easier to implement in the real world, as these controllers could be less costly in terms of hardware, and faster and more energy-efficient in processing. Additionally, symbolic formulae and graphs may offer an implicit feature selection advantage, potentially reducing the number of sensors required in the modules. Still, the extent to which different controller types would be affected by the reality gap remains uncertain and should be investigated

further, at least through sim-to-sim experimentation.

Research Question 3.2 *Can diversity foster the optimization of MSRs that are interpretable, effective, and adaptable? How important is it to diversify in terms of body designs, controllers features, and robot behaviors?*

We provided a positive answer to this question in Chapter 12, where we promoted diversity in MSRs across three key axes, resulting in significant improvements in performance and adaptability, even when using interpretable controllers. Additionally, we observed parallels with biodiversity, particularly regarding the most critical axes of diversity.

In Chapter 12, we explored the scenario of body-brain co-optimization for MSRs, using two controller representations—ANNs and graphs—both optimized with EC for a locomotion task. We aimed to mimic biological diversity through a QD framework that diversified MSRs across three key axes: brains, bodies, and behaviors. Our experiments evaluated both the performance of the optimized MSRs and their resulting diversity and adaptability to similar tasks, with a specific focus on the interpretability of the resulting controllers.

Our findings demonstrated comparable performance between graph-based controllers and ANNs when optimized using QD, though the search process for this technique was slower compared to greedier optimization approaches. We also observed notable diversity, particularly in terms of body morphologies and behaviors, with similarities across both controller representations. Importantly, we found that diversity significantly enhanced adaptability, especially with respect to body and behavior traits, underscoring their critical role in achieving robust, high-performing MSRs.

When transitioning to hardware implementations, we expect that the diversity of optimized MSRs, along with a more compact controller representation, could offer an advantage by increasing the likelihood of finding transferable solutions. However, selecting the most promising designs for real-world implementation would remain a complex task. Future work should focus on identifying the traits that make a MSR more or less transferable and either optimize directly for these traits [196, 197] or incorporate them as descriptors within a QD framework.

Another challenge of this study concerns the adaptability experiments, which would be difficult to replicate in real-world settings without significant human intervention. Testing a wide range of optimized robots in a new scenario would require changes not only to their controllers—which might be possible just through software updates—but also the physical re-assembly of their modules. While such reconfiguration could theoretically be automated with external infrastructure [314] or through self-reconfiguring modules [368], the feasibility of minimizing external intervention remains uncertain. A potential solution would be

to conduct an initial selection phase in simulation—if the new scenario could be adequately simulated—and only transfer the most promising designs for testing in real-world environments.

13.2 Impact and Future Perspectives

The most immediate question to ask at the conclusion of a Ph.D. thesis is: "what new knowledge or contribution has emerged that was not there before?". With this thesis, we have gained valuable insights and learned important lessons:

1. *Bio-inspiration can enhance the performance of embodied agents*: while AI models do not need to replicate biological intelligence perfectly, incorporating certain principles from biology (e.g., ANNs with internal states or learning capabilities) can significantly improve performance.

2. *Interpretability does not need to compromise performance*: simple metrics, such as model size, can be used to quantify and foster interpretability. In this regard, Genetic Programming (GP) proves effective for optimizing interpretable models without sacrificing performance.

3. *Bio-inspiration and interpretability are not mutually exclusive*: in fact, we can strike a balance, integrating key aspects of both, such as QD optimization and GP, to develop models that are high-performing and trustworthy.

In addition, when combined with the work of more applied roboticists, this foundational research could lead to significant future advancements also in the field of robotics.

13.2.1 Impacts on Application Domains

There are several potential application domains where MSRs could have a positive impact. The first domain involves *exploration in remote and inaccessible areas* (e.g., space exploration [270, 313]), where these robots could be deployed for data collection and even reassembled as needed. In this case, adaptability is crucial, likely more so than interpretability, since these robots would have minimal direct interaction with humans. This scenario presents an opportunity to exploit findings from biological mimicry, such as using memory or plasticity in controllers to enhance robustness.

Second, these robots could be scaled down to the nanoscale for *targeted drug delivery* [6, 180, 27]. Here, bio-mimicry would play a critical role in ensuring seamless integration into biological organisms. However, due to the sensitivity of such applications, incorporating interpretable controllers would be essential to

understand the robots inner workings and to enable corrective actions if necessary.

Third, another promising area concerns *prosthetics*. Designed with muscle tissue in mind, these robots could serve as ideal candidates for muscle replacement in artificial limbs—recent studies have made progress in soft robotics for hand [281] and foot prosthetics [336]. Integrating bio-inspired control techniques could further advance developments in this area, ultimately benefiting disabled patients.

Finally, these robots could be utilized to implement modules within *humanoid robots*. Current research is indeed advancing toward humanoid robots with more natural human-like gait patterns. A notable example is the Adam humanoid robot by PNDbotics [476], which uses an end-to-end ANN trained with RL and imitation learning from human data to control its gait[1]. This approach results in smoother movements compared to robots using more traditional control systems, such as those developed by Boston Dynamics. MSRs could be integrated into these robots, acting as artificial muscles to enable even smoother movements and improve compliance and adaptability.

We remark that a significant challenge in enabling these applications lies in bridging the "reality gap" between simulation and real-world robotics. As highlighted in earlier sections, the studies and evaluations presented in this thesis were conducted exclusively in simulation, primarily due to the absence of a high-fidelity physical counterpart for MSRs. To facilitate future applications, we envision leveraging well-established techniques to address this reality gap [372], such as incorporating additional physical constraints into the simulator, simulating environmental noise, or experimenting with sim-to-sim transfer methods. Ultimately, the findings should be validated on physical robots. A practical initial approach might involve working with simpler variants of modular robots that are rigid rather than soft, as these are easier to control and can be readily fabricated using 3D printing technologies [165].

13.2.2 Future Perspective: a Hierarchical Approach to Merge Interpretability and Biological Fidelity

Throughout this thesis, we aimed to merge biological inspiration with interpretability in MSRs, though much of our focus ultimately involved replacing biologically inspired controllers with more interpretable ones. In this final section, we propose an alternative paradigm that allows for the coexistence of both, which we intend to explore as a future research direction. This new approach is based on a *hierarchical control model* for MSRs and draws strong inspiration from decision-making processes in nature.

[1] Video at `https://youtu.be/pzxB-va0qNo?si=g14hcV1DFIR48THq`.

To illustrate it, consider a scenario where a person or animal must make a decision—such as choosing a path to travel from point A to point B. Several factors, such as difficulty, distance, and even the surrounding views, are weighed before reaching a decision. However, when it comes to carrying out the decision, such as moving their legs, the specifics of leg movement are typically instinctive. In this process, we can distinguish two layers: the higher layer is the decision-making process, while the lower layer is the instinctive action needed to execute the decision.

Translating this model to a robotic context, we aim for interpretability at the higher decision-making level—for example, understanding why the robot chooses to go left rather than right. However, we are less concerned with the mechanics of how the movement is executed, as long as the robot successfully completes its task. From an implementation standpoint, this means leveraging biological fidelity to create functional modules that can effectively and robustly carry out sub-actions. These modules could adapt to small environmental changes (e.g., small obstacles) using SNNs or Hebbian learning. At a higher level, interpretable controllers would oversee decision-making, determining which tasks need to be performed and instructing the lower-level, biologically inspired modules accordingly.

Glossary

3B-QD Body-Brain-Behavior Quality-Diversity.

AI Artificial Intelligence.

AL Active Learning.

aMSE Average Mean Squared Error.

ANN Artificial Neural Network.

CA Cellular Automaton.

CGP Cartesian Genetic Programming.

CNN Convolutional Neural Network.

CPG Central Pattern Generator.

CVP Contribution Variance Pruning.

CVT Centroidal Voronoi Tessellation.

DAG Directed Acyclic Graph.

DT Decision Tree.

EA Evolutionary Algorithm.

EANT Evolutionary Acquisition of Neural Topologies.

EC Evolutionary Computation.

ER Evolutionary Robotics.

ES Evolutionary Strategy.

EvoGym Evolution Gym.

FFT Fast Fourier Transform.

G-QD Graph Quality-Diversity.

GA Genetic Algorithm.

GE Grammatical Evolution.

GGP Graph-based Genetic Programming.

GP Genetic Programming.

GUI Graphical User Interface.

H-MLP Hebbian MLP.

HV HyperVolume.

IRL Interpretable Reinforcement Learning.

LCS Learning Classifier System.

LGP Linear Genetic Programming.

LIF Leaky Integrate and Fire.

LLM Large Language Model.

LMP Least-Magnitude Pruning.

LSTM Long Short Term Memory.

MARL Multi-Agent Reinforcement Learning.

MAS Multi-Agent System.

MDP Markov Decision Process.

ME MAP-Elites.

ML Machine Learning.

ML-PIE Model Learning with Personalized Interpretability Estimation.

MLP Multi-Layer Perceptron.

MSE Mean Squared Error.

MSR Modular Soft Robot.

NCA Neural Cellular Automata.

NE Neuroevolution.

NEAT NeuroEvolution of Augmenting Topologies.

NLP Natural Language Processing.

NS Novelty Search.

NSGA-II Non-dominated Sorting Genetic Algorithm II.

PCA Principal Component Analysis.

PHI Proxy of Human Interpretability.

PPO Proximal Policy Optimization.

QD Quality-Diversity.

ReLU Rectified Linear Unit.

RL Reinforcement Learning.

RNN Recurrent Neural Network.

SAC Soft Actor Critic.

SGD Stochastic Gradient Descent.

SiLU Sigmoid Linear Unit.

SNCA Spiking Neural Cellular Automata.

SNN Spiking Neural Network.

SNN-H SNN with Homeostasis.

SR Symbolic Regression.

STDP Spike-Timing-Dependent Plasticity.

TORCS the Open Racing Car Simulator.

TPG Tangled Program Graph.

VSR Voxel-based Soft Robot.

XAI Explainable Artificial Intelligence.

XRL Explainable Reinforcement Learning.

Bibliography

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] A. Adadi and M. Berrada. Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). *IEEE access*, 6:52138–52160, 2018.

[3] H. Aerts, W. Fias, K. Caeyenberghs, and D. Marinazzo. Brain networks under attack: robustness properties and the impact of lesions. *Brain*, 139 (12):3063–3083, 2016.

[4] K. Akinci and A. Philippides. Evolving recurrent neural network controllers by incremental fitness shaping. In *Artificial Life Conference Proceedings*, pages 416–423, 2019.

[5] R. J. Alattas, S. Patel, and T. M. Sobh. Evolutionary modular robotics: Survey and analysis. *Journal of Intelligent & Robotic Systems*, 95(3):815–828, 2019.

[6] G. Alici. Towards soft robotic devices for site-specific drug delivery. *Expert review of medical devices*, 12(6):703–715, 2015.

[7] R. Amaral, A. Ianta, C. Bayer, R. J. Smith, and M. I. Heywood. Benchmarking genetic programming in a multi-action reinforcement learning locomotion task. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 522–525, 2022.

[8] S. Anjomshoae, A. Najjar, D. Calvaresi, and K. Främling. Explainable agents and robots: Results from a systematic literature review. In *18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019*, pages 1078–1088. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[9] A. Ansuini, E. Medvet, F. A. Pellegrino, and M. Zullich. Investigating Similarity Metrics for Convolutional Neural Networks in the Case of Unstructured Pruning. In *International Conference on Pattern Recognition Applications and Methods*, pages 87–111. Springer, 2020.

[10] A. Ansuini, E. Medvet, F. A. Pellegrino, and M. Zullich. On the Similarity between Hidden Layers of Pruned and Unpruned Convolutional Neural Networks. In *ICPRAM*, pages 52–59, 2020.

[11] S. Anwar, K. Hwang, and W. Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):1–18, 2017.

[12] S. Aradi. Survey of deep reinforcement learning for motion planning of autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(2):740–759, 2020.

[13] C. Arcuri, C. Mecca, R. Bianchi, I. Giambanco, and R. Donato. The pathophysiological role of microglia in dynamic surveillance, phagocytosis and structural remodeling of the developing CNS. *Frontiers in Molecular Neuroscience*, 10:191, 2017.

[14] I. Asimov. *I, robot*, volume 1. Spectra, 1950.

[15] J. Auerbach and J. C. Bongard. Evolution of functional specialization in a morphologically homogeneous robot. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 89–96, 2009.

[16] J. E. Auerbach and J. C. Bongard. Environmental influence on the evolution of morphological complexity in machines. *PLoS Comput Biol*, 10(1): e1003399, 2014.

[17] D. Auge, J. Hille, E. Mueller, and A. Knoll. A survey of encoding techniques for signal processing in spiking neural networks. *Neural Processing Letters*, 53(6):4693–4710, 2021.

[18] J. Bacardit, A. E. Brownlee, S. Cagnoni, G. Iacca, J. McCall, and D. Walker. The intersection of evolutionary computation and explainable AI. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1757–1762, 2022.

[19] J. M. Baldwin. A new factor in evolution. *American naturalist*, pages 536–553, 2018.

[20] H. Banerjee, Z. T. H. Tse, and H. Ren. Soft robotics with compliance and adaptation for biomedical applications and forthcoming challenges. *Int. J. Robot. Autom*, 33(1):69–80, 2018.

[21] B. R. Bartoldson, A. S. Morcos, A. Barbu, and G. Erlebacher. The generalization-stability tradeoff in neural network pruning. *arXiv preprint arXiv:1906.03728*, 2019.

[22] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Inference of regular expressions for text extraction from examples. *IEEE Transactions on Knowledge and Data Engineering*, 28(5):1217–1230, 2016.

[23] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Active learning of regular expressions for entity extraction. *IEEE Transactions on Cybernetics*, 48(3):1067–1080, 2017.

[24] A. Bartoli, A. De Lorenzo, E. Medvet, and G. Squillero. Multi-level diversity promotion strategies for Grammar-guided Genetic Programming. *Applied Soft Computing*, 83:105599, 2019.

[25] D. S. Bassett and O. Sporns. Network neuroscience. *Nature neuroscience*, 20(3):353–364, 2017.

[26] J. A. Batsis and S. Buscemi. Sarcopenia, sarcopenic obesity and insulin resistance. In *Medical Complications of Type 2 Diabetes*. IntechOpen, 2011.

[27] R. Beatty, K. L. Mendez, L. H. Schreiber, R. Tarpey, W. Whyte, Y. Fan, S. T. Robinson, J. O'Dwyer, A. J. Simpkin, J. Tannian, et al. Soft robot–mediated autonomous adaptation to fibrotic capsule formation for improved drug delivery. *Science Robotics*, 8(81):eabq4821, 2023.

[28] G. A. Bekey and K. Y. Goldberg. *Neural networks in robotics*, volume 202. Springer Science & Business Media, 2012.

[29] R. Bellman, I. Glicksberg, and O. Gross. On the "bang-bang" control problem. *Quarterly of Applied Mathematics*, 14(1):11–18, 1956.

[30] Y. Bengio, N. Le Roux, P. Vincent, O. Delalleau, and P. Marcotte. Convex neural networks. *Advances in neural information processing systems*, 18: 123, 2006.

[31] M. Benk and A. Ferrario. Explaining Interpretable Machine Learning: Theory, Methods and Applications. *Methods and Applications (December 11, 2020)*, 2020.

[32] J. Bhatia, H. Jackson, Y. Tian, J. Xu, and W. Matusik. Evolution Gym: A Large-Scale Benchmark for Evolving Soft Robots. *Advances in Neural Information Processing Systems*, 34, 2021.

[33] Z. Bing, C. Meschede, F. Röhrbein, K. Huang, and A. C. Knoll. A survey of robotics control based on learning-inspired spiking neural networks. *Frontiers in neurorobotics*, 12:35, 2018.

[34] C. M. Bishop. *Neural Networks for Pattern Recognition*, chapter 9.5.3 - Saliency of Weights. Clarendon Press, 1995.

[35] J. Blank and K. Deb. pymoo: Multi-Objective Optimization in Python. *IEEE Access*, 8:89497–89509, 2020.

[36] P. Blouw, X. Choo, E. Hunsberger, and C. Eliasmith. Benchmarking keyword spotting efficiency on neuromorphic hardware. In *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, pages 1–8, 2019.

[37] N. A. Bomberger, A. M. Waxman, B. J. Rhodes, and N. A. Sheldon. A new approach to higher-level information fusion using associative learning in semantic networks of spiking neurons. *Information Fusion*, 8(3):227–251, 2007.

[38] J. Bongard. Morphological change in machines accelerates the evolution of robust behavior. *Proceedings of the National Academy of Sciences*, 108 (4):1234–1239, 2011.

[39] J. C. Bongard. Evolutionary robotics. *Communications of the ACM*, 56(8): 74–83, 2013.

[40] J. Bono, S. Zannone, V. Pedrosa, and C. Clopath. Learning predictive cognitive maps with spiking neurons during behavior and replays. *Elife*, 12:e80671, 2023.

[41] C. Bordier, C. Nicolini, and A. Bifone. Graph analysis and modularity of brain functional connectivity networks: searching for the optimal threshold. *Frontiers in neuroscience*, 11:441, 2017.

[42] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne. Spiking neural networks hardware implementations and challenges: A survey. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 15(2):1–35, 2019.

[43] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, et al. JAX: composable transformations of Python+ NumPy programs, 2018.

[44] M. Brameier, W. Banzhaf, and W. Banzhaf. *Linear genetic programming*, volume 1. Springer, 2007.

[45] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984.

[46] L. A. Breslow and D. W. Aha. Simplifying decision trees: A survey. *The Knowledge Engineering Review*, 12(01):1–40, 1997.

[47] K. Brotto Rebuli, M. Giacobini, S. Silva, and L. Vanneschi. A Comparison of Structural Complexity Metrics for Explainable Genetic Programming. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, pages 539–542, 2023.

[48] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language Models are Few-Shot Learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`.

[49] T. H. Brown, E. W. Kairiss, and C. L. Keenan. Hebbian synapses: biophysical mechanisms and algorithms. *Annual review of neuroscience*, 13(1): 475–511, 1990.

[50] N. Brunel. Hebbian learning of context in recurrent neural networks. *Neural computation*, 8(8):1677–1710, 1996.

[51] A. Brunete, A. Ranganath, S. Segovia, J. P. De Frutos, M. Hernando, and E. Gambao. Current trends in reconfigurable modular robots design. *International Journal of Advanced Robotic Systems*, 14(3):1729881417710457, 2017.

[52] W. Burger, M. J. Burge, M. J. Burge, and M. J. Burge. *Principles of digital image processing*, volume 111. Springer, 2009.

[53] N. F. Butte, C. Garza, and M. de Onis. Evaluation of the feasibility of international growth standards for school-aged children and adolescents. *The Journal of nutrition*, 137(1):153–157, 2007.

[54] L. Cai, Z. An, C. Yang, and Y. Xu. Softer Pruning, Incremental Regularization. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 224–230, 2021. doi: 10.1109/ICPR48806.2021.9412993.

[55] A. Cano, A. Zafra, and S. Ventura. An interpretable classification rule mining algorithm. *Information Sciences*, 240:1–20, 2013.

[56] K. Čapek. *RUR*. Standard Ebooks, 2020.

[57] N. Caporale and Y. Dan. Spike timing–dependent plasticity: a Hebbian learning rule. *Annu. Rev. Neurosci.*, 31:25–46, 2008.

[58] L. Cazenille. Ensemble feature extraction for multi-container quality-diversity algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 75–83, 2021.

[59] F. Chalumeau, B. Lim, R. Boige, M. Allard, L. Grillotti, M. Flageat, V. Macé, A. Flajolet, T. Pierrot, and A. Cully. QDax: A Library for Quality-Diversity and Population-based Algorithms with Hardware Acceleration. *arXiv preprint arXiv:2308.03665*, 2023.

[60] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 167–174. ACM, 2013.

[61] N. Cheney, J. Clune, and H. Lipson. Evolved electrophysiological soft robots. In *Artificial Life Conference Proceedings 14*, pages 222–229. MIT Press, 2014.

[62] N. Cheney, J. Bongard, and H. Lipson. Evolving soft robots in tight spaces. In *Proceedings of the 2015 annual conference on Genetic and Evolutionary Computation*, pages 935–942. ACM, 2015.

[63] R. Chrisley. Embodied artificial intelligence. *Artificial intelligence*, 149(1): 131–150, 2003.

[64] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[65] F. Corucci, N. Cheney, F. Giorgio-Serchi, J. Bongard, and C. Laschi. Evolving soft locomotion in aquatic and terrestrial environments: effects of material properties and environmental transitions. *Soft robotics*, 5(4):475–495, 2018.

[66] R. Coulom. *Reinforcement learning using neural networks, with applications to motor control*. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 2002.

[67] S. Coyle, C. Majidi, P. LeDuc, and K. J. Hsia. Bio-inspired soft robotics: Material selection, actuation, and design. *Extreme Mechanics Letters*, 22: 51–59, 2018.

[68] A. Cully. Autonomous skill discovery with quality-diversity and unsupervised descriptors. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 81–89, 2019.

[69] A. Cully and Y. Demiris. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22 (2):245–259, 2017.

[70] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

[71] L. L. Custode and G. Iacca. Evolutionary learning of interpretable decision trees. *arXiv preprint arXiv:2012.07723*, 2020.

[72] L. L. Custode and G. Iacca. A co-evolutionary approach to interpretable reinforcement learning in environments with continuous action spaces. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2021.

[73] L. L. Custode and G. Iacca. Interpretable pipelines with evolutionary optimized modules for reinforcement learning tasks with visual inputs. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 224–227, 2022.

[74] L. L. Custode and G. Iacca. Interpretable AI for policy-making in pandemics. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1763–1769, 2022.

[75] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[76] J. Daly, D. Casper, M. Farooq, A. James, A. Khan, P. Mulgrew, D. Tyebkhan, B. Vo, and J. Rieffel. Robustness for Free: Quality-Diversity Driven Discovery of Agile Soft Robotic Gaits. *arXiv preprint arXiv:2311.01245*, 2023.

[77] L. Damiano, P. Stano, and P. Stano. Synthetic Biology and Artificial Intelligence. Grounding a cross-disciplinary approach to the synthetic exploration of (embodied) cognition. *Complex Syst*, 27:199–228, 2018.

[78] C. Darwin and W. F. Bynum. *The origin of species by means of natural selection: or, the preservation of favored races in the struggle for life*. AL Burt New York, 2009.

[79] A. Das and P. Rad. Opportunities and challenges in explainable artificial intelligence (xai): A survey. *arXiv preprint arXiv:2006.11371*, 2020.

[80] J. Daudelin, G. Jing, T. Tosun, M. Yim, H. Kress-Gazit, and M. Campbell. An integrated system for perception-driven autonomy with modular robots. *Science Robotics*, 3(23):eaat4983, 2018.

[81] Q. T. Davis, S. Woodman, M. Landesberg, R. Kramer-Bottiglio, and J. Bongard. Subtract to adapt: Autotomic robots. In *2023 IEEE International Conference on Soft Robotics (RoboSoft)*, pages 1–6. IEEE, 2023.

[82] M. De Carlo, D. Zeeuwe, E. Ferrante, G. Meynen, J. Ellers, and A. Eiben. Influences of Artificial Speciation on Morphological Robot Evolution. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 2272–2279. IEEE, 2020.

[83] J. De Freitas, G. L. Pappa, A. S. da Silva, M. A. Gonc, E. Moura, A. Veloso, A. H. Laender, M. G. de Carvalho, et al. Active learning genetic programming for record deduplication. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2010.

[84] K. A. De Jong. *Evolutionary Computation: A Unified Approach*. MIT Press, 2006.

[85] K. Deb. Multi-objective genetic algorithm: problem difficulties and construction of test functions. *Evolutionary Computation, I*, 999(7):205–230, 1998.

[86] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[87] C. Della Santina, R. K. Katzschmann, A. Biechi, and D. Rus. Dynamic control of soft robots interacting with the environment. In *2018 IEEE International Conference on Soft Robotics (RoboSoft)*, pages 46–53. IEEE, 2018.

[88] S. Denève, A. Alemi, and R. Bourdoukan. The brain as an efficient and robust adaptive learner. *Neuron*, 94(5):969–977, 2017.

[89] P. Diaconis and R. L. Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39 (2):262–268, 1977.

[90] M. Divband Soorati and H. Hamann. The Effect of Fitness Function Design on Performance in Evolutionary Robotics: The Influence of a Priori Knowledge. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, page 153–160, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450334723. doi: 10.1145/2739480.2754676. URL https://doi.org/10.1145/2739480.2754676.

[91] K. Dobs, J. Martinez, A. J. E. Kell, and N. Kanwisher. Brain-like functional specialization emerges spontaneously in deep neural networks. *Science Advances*, 8(11):eabl8913, 2022.

[92] S. Doncieux and A. Coninx. Open-ended evolution with multi-containers QD. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 107–108, 2018.

[93] S. Doncieux and J.-B. Mouret. Behavioral diversity measures for evolutionary robotics. In *IEEE congress on evolutionary computation*, pages 1–8, 2010. doi: 10.1109/CEC.2010.5586100.

[94] S. Doncieux, N. Bredeche, J.-B. Mouret, and A. E. Eiben. Evolutionary robotics: what, why, and where to. *Frontiers in Robotics and AI*, 2:4, 2015.

[95] D. Doran, S. Schulz, and T. R. Besold. What does explainable AI really mean? A new conceptualization of perspectives. *CoRR*, abs/1710.00794, 2017. URL http://arxiv.org/abs/1710.00794.

[96] R. Doursat, H. Sayama, and O. Michel. *Morphogenetic engineering: toward programmable complex systems*. Springer, 2012.

[97] F. K. Došilović, M. Brčić, and N. Hlupić. Explainable artificial intelligence: A survey. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 0210–0215, 2018. doi: 10.23919/MIPRO.2018.8400040.

[98] B. Dresp-Langley. From biological synapses to “intelligent” robots. *Electronics*, 11(5):707, 2022.

[99] A. Eiben and E. Hart. If it evolves it needs to learn. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pages 1383–1384, 2020.

[100] A. E. Eiben, J. E. Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[101] A. Ekart and S. Z. Nemeth. Selection based on the pareto nondomination criterion for controlling code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 2(1):61–73, 2001.

[102] M. F. A. R. D. T. (FAIR)†, A. Bakhtin, N. Brown, E. Dinan, G. Farina, C. Flaherty, D. Fried, A. Goff, J. Gray, H. Hu, et al. Human-level play in the game of Diplomacy by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074, 2022.

[103] F.-L. Fan, J. Xiong, M. Li, and G. Wang. On interpretability of artificial neural networks: A survey. *IEEE Transactions on Radiation and Plasma Medical Sciences*, 5(6):741–760, 2021.

[104] W. Fedus, B. Zoph, and N. Shazeer. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

[105] F. H. Fenton, E. M. Cherry, A. Karma, and W.-J. Rappel. Modeling wave propagation in realistic heart geometries using the phase-field method. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 15(1), 2005.

[106] A. Ferigo and G. Iacca. Self-Building Neural Networks. In *Conference on Genetic and Evolutionary Computation Companion*, GECCO ’23, page 643–646, New York, NY, USA, 2023. ACM. ISBN 9798400701207.

[107] A. Ferigo, G. Iacca, and E. Medvet. Beyond Body Shape and Brain: Evolving the Sensory Apparatus of Voxel-based Soft Robots. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer, 2021.

[108] A. Ferigo, L. L. Custode, and G. Iacca. Quality Diversity Evolutionary Learning of Decision Trees. *arXiv preprint arXiv:2208.12758*, 2022.

[109] A. Ferigo, G. Iacca, E. Medvet, and F. Pigozzi. Evolving Hebbian learning rules in voxel-based soft robots. *IEEE Transactions on Cognitive and Developmental Systems*, 2022.

[110] A. Ferigo, L. Soros, E. Medvet, and G. Iacca. On the Entanglement between Evolvability and Fitness: an Experimental Study on Voxel-based Soft Robots. In *ALIFE: PROCEEDINGS OF THE ARTIFICIAL LIFE CONFERENCE*. MIT press, 2022.

[111] A. Ferigo, L. L. Custode, and G. Iacca. Quality–diversity optimization of decision trees for interpretable reinforcement learning. *Neural Computing and Applications*, pages 1–12, 2023.

[112] A. Ferigo, E. Cunegatti, and G. Iacca. Neuron-centric Hebbian Learning. *arXiv preprint arXiv:2403.12076*, 2024.

[113] A. Ferigo, G. Iacca, E. Medvet, and G. Nadizar. Totipotent Neural Controllers for Modular Soft Robots: Achieving Specialization in Body-Brain Co-Evolution through Hebbian Learning. *Neurocomputing*, page 128811, 2024.

[114] A. Fernandez, F. Herrera, O. Cordon, M. J. del Jesus, and F. Marcelloni. Evolutionary fuzzy systems for explainable artificial intelligence: Why, when, what for, and where to? *IEEE Computational intelligence magazine*, 14(1):69–81, 2019.

[115] R. P. Feynman, R. B. Leighton, and M. Sands. The feynman lectures on physics; vol. i. *American Journal of Physics*, 33(9):750–752, 1965.

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

[117] L. Françoso Dal Piccol Sotto, P. Kaufmann, T. Atkinson, R. Kalkreuth, and M. Porto Basgalupp. Graph representations in genetic programming. *Genetic Programming and Evolvable Machines*, 22(4):607–636, 2021.

[118] J. Frankle and M. Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJl-b3RcF7`.

[119] S. Franklin. Autonomous agents as embodied AI. *Cybernetics & Systems*, 28(6):499–520, 1997.

[120] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem. Brax–A Differentiable Physics Engine for Large Scale Rigid Body Simulation. *arXiv preprint arXiv:2106.13281*, 2021.

[121] A. L. Freire, G. A. Barreto, M. Veloso, and A. T. Varela. Short-term memory mechanisms in neural network learning of robot navigation tasks: A case

study. In *2009 6th Latin American robotics symposium (LARS 2009)*, pages 1–6. IEEE, 2009.

[122] A. A. Freitas. Comprehensible classification models: a position paper. *ACM SIGKDD explorations newsletter*, 15(1):1–10, 2014.

[123] G. Fusco and A. Minelli. Phenotypic plasticity in development and evolution: facts and concepts, 2010.

[124] A. Gaier, A. Asteroth, and J.-B. Mouret. Are quality diversity algorithms better at generating stepping stones than objective-based search? In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 115–116, 2019.

[125] W. Gerstner and W. M. Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[126] R. C. Gerum, A. Erpenbeck, P. Krauss, and A. Schilling. Sparsity through evolutionary pruning prevents neuronal networks from overfitting. *Neural Networks*, 128:305–312, 2020.

[127] C. L. Giles and C. W. Omlin. Pruning recurrent neural networks for improved generalization performance. *IEEE transactions on neural networks*, 5(5):848–851, 1994.

[128] C. Glanois, P. Weng, M. Zimmer, D. Li, T. Yang, J. Hao, and W. Liu. A survey on interpretable reinforcement learning. *arXiv preprint arXiv:2112.13112*, 2021.

[129] A. Glass, D. L. McGuinness, and M. Wolverton. Toward establishing trust in adaptive agents. In *Proceedings of the 13th international conference on Intelligent user interfaces*, pages 227–236, 2008.

[130] R. Goebel, A. Chander, K. Holzinger, F. Lecue, Z. Akata, S. Stumpf, P. Kieseberg, and A. Holzinger. Explainable ai: The new 42? In A. Holzinger, P. Kieseberg, A. M. Tjoa, and E. Weippl, editors, *Machine Learning and Knowledge Extraction*, pages 295–303, Cham, 2018. Springer International Publishing. ISBN 978-3-319-99740-7.

[131] D. Gravina, A. Liapis, and G. N. Yannakakis. Fusing novelty and surprise for evolving robot morphologies. In *Proceedings of the genetic and evolutionary computation conference*, pages 93–100, 2018.

[132] D. Gravina, A. Liapis, and G. N. Yannakakis. Blending notions of diversity for MAP-Elites. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 117–118, 2019.

[133] L. Grillotti and A. Cully. Kheperax: a lightweight jax-based robot control environment for benchmarking quality-diversity algorithms. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, pages 2163–2165, 2023.

[134] R. Guidotti, A. Monreale, S. Ruggieri, D. Pedreschi, F. Turini, and F. Giannotti. Local rule-based explanations of black box decision systems. *arXiv preprint arXiv:1805.10820*, 2018.

[135] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):1–42, 2018.

[136] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

[137] W. Guo, M. E. Fouda, A. M. Eltawil, and K. N. Salama. Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems. *Frontiers in Neuroscience*, 15:638474, 2021.

[138] M. Gupta, A. Ambikapathi, and S. Ramasamy. Hebbnet: A simplified hebbian learning framework to do biologically plausible learning. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3115–3119. IEEE, 2021.

[139] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[140] H. Hagras. Toward human-understandable, explainable AI. *Computer*, 51 (9):28–36, 2018.

[141] A. Hallawa, G. Iacca, C. Sariman, T. Rahman, M. Cochez, and G. Ascheid. Morphological evolution for pipe inspection using Robot Operating System (ROS). *Materials and Manufacturing Processes*, 35(6):714–724, 2020.

[142] H. Hamann. *Swarm robotics: A formal approach*, volume 221. Springer, 2018.

[143] S. Han, J. Pool, J. Tran, and W. Dally. Learning both Weights and Connections for Efficient Neural Network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1135–1143. Curran Associates, Inc., 2015.

[144] D. Harrison Jr and D. L. Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management*, 5(1):81–102, 1978.

[145] E. Hart and L. K. Le Goff. Artificial evolution of robot bodies and control: on the interaction between evolution, learning and culture. *Philosophical Transactions of the Royal Society B*, 377(1843):20210117, 2022.

[146] J. J. Hatherley. Limits of trust in medical AI. *Journal of medical ethics*, 46 (7):478–481, 2020.

[147] H. Hauser, A. J. Ijspeert, R. M. Füchslin, R. Pfeifer, and W. Maass. Towards a theoretical foundation for morphological computation with compliant bodies. *Biological Cybernetics*, 105(5):355–370, Dec 2011. ISSN 1432-0770. doi: 10.1007/s00422-012-0471-0. URL `https://doi.org/10.1007/s00422-012-0471-0`.

[148] W. He, T.-Y. Lee, J. van Baar, K. Wittenburg, and H.-W. Shen. Dynamicsexplorer: Visual analytics for robot control tasks involving dynamics and lstm-based control policies. In *2020 IEEE Pacific Visualization Symposium (PacificVis)*, pages 36–45. IEEE, 2020.

[149] D. O. Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[150] D. Hein, S. Udluft, and T. A. Runkler. Interpretable policies for reinforcement learning by genetic programming. *Engineering Applications of Artificial Intelligence*, 76:158–169, 2018.

[151] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. *Proceedings of the AAAI conference on artificial intelligence*, 32(1), 2018.

[152] S. Herculano-Houzel. The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost. *Proceedings of the National Academy of Sciences*, 109(Supplement 1):10661–10668, 2012.

[153] J. Hiller and H. Lipson. Automatic design and manufacture of soft robots. *IEEE Transactions on Robotics*, 28(2):457–466, 2011.

[154] J. Hiller and H. Lipson. Dynamic Simulation of Soft Multimaterial 3D-printed Objects. *Soft Robotics*, 1(1):88–101, 2014.

[155] G. E. Hinton, S. J. Nowlan, et al. How learning can guide evolution. *Adaptive individuals in evolving populations: models and algorithms*, 26: 447–454, 1996.

[156] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste. Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. *arXiv preprint arXiv:2102.00554*, 2021.

[157] R. R. Hoffman, S. T. Mueller, G. Klein, and J. Litman. Metrics for explainable AI: Challenges and prospects. *arXiv preprint arXiv:1812.04608*, 2018.

[158] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[159] A. Holzinger. From Machine Learning to Explainable AI. In *2018 World Symposium on Digital Intelligence for Systems and Machines (DISA)*, pages 55–66, 2018. doi: 10.1109/DISA.2018.8490530.

[160] K. Horibe, K. Walker, and S. Risi. Regenerating Soft Robots Through Neural Cellular Automata. In *EuroGP*, pages 36–50, 2021.

[161] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[162] B.-Y. Hu, J. P. Weick, J. Yu, L.-X. Ma, X.-Q. Zhang, J. A. Thomson, and S.-C. Zhang. Neural differentiation of human induced pluripotent stem cells follows developmental principles but with variable potency. *Proceedings of the National Academy of Sciences*, 107(9):4335–4340, 2010.

[163] W. Huang, I. Mordatch, and D. Pathak. One policy to control them all: Shared modular policies for agent-agnostic control. In *International Conference on Machine Learning*, pages 4455–4464. PMLR, 2020.

[164] D. C. Hughes, S. Ellefsen, and K. Baar. Adaptations to endurance and strength training. *Cold Spring Harbor perspectives in medicine*, 8(6): a029769, 2018.

[165] E. Hupkes, M. Jelisavcic, and A. E. Eiben. Revolve: a versatile simulator for online robot evolution. In *Applications of Evolutionary Computation: 21st International Conference, EvoApplications 2018, Parma, Italy, April 4-6, 2018, Proceedings 21*, pages 687–702. Springer, 2018.

[166] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 51(1):141–154, 2011.

[167] J. Iglesias, J. Eriksson, F. Grize, M. Tomassini, and A. E. Villa. Dynamics of pruning in simulated large-scale spiking neural networks. *Biosystems*, 79 (1-3):11–20, 2005.

[168] A. J. Ijspeert. Central pattern generators for locomotion control in animals and robots: a review. *Neural networks*, 21(4):642–653, 2008.

[169] R. Isele and C. Bizer. Active learning of expressive linkage rules using genetic programming. *Journal of web semantics*, 23:2–15, 2013.

[170] E. M. Izhikevich. Which model to use for cortical spiking neurons? *IEEE transactions on neural networks*, 15(5):1063–1070, 2004.

[171] Y. Izza, A. Ignatiev, and J. Marques-Silva. On explaining decision trees. *arXiv preprint arXiv:2010.11034*, 2020.

[172] S. M. J. Jalali, S. Ahmadian, A. Khosravi, S. Mirjalili, M. R. Mahmoudi, and S. Nahavandi. Neuroevolution-based autonomous robot navigation: a comparative study. *Cognitive Systems Research*, 62:35–43, 2020.

[173] N. Javed, F. R. Gobet, and P. Lane. Simplification of genetic programs: a literature survey. *Data Min. Knowl. Discov.*, 36:1279–1300, 2022.

[174] L. Jin, S. Li, J. Yu, and J. He. Robot manipulator control using neural networks: A survey. *Neurocomputing*, 285:23–34, 2018.

[175] Y. Jin and Y. Meng. Morphogenetic Robotics: An Emerging New Field in Developmental Robotics. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(2):145–160, 2011.

[176] A. Jobin, M. Ienca, and E. Vayena. The global landscape of AI ethics guidelines. *Nature Machine Intelligence*, 1, 09 2019. doi: 10.1038/s42256-019-0088-2.

[177] M. H. Johnson. Functional brain development in humans. *Nature Reviews Neuroscience*, 2(7):475–483, 2001.

[178] J. H. Jones. Primates and the evolution of long, slow life histories. *Current Biology*, 21(18):R708–R717, 2011.

[179] J. Jordan, M. Schmidt, W. Senn, and M. A. Petrovici. Evolving interpretable plasticity for spiking networks. *Elife*, 10, 2021.

[180] E. B. Joyee and Y. Pan. Additive manufacturing of multi-material soft robot for on-demand drug delivery applications. *Journal of Manufacturing Processes*, 56:1178–1184, 2020.

[181] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[182] Y. Karayiannidis, L. Droukas, D. Papageorgiou, and Z. Doulgeri. Robot control for task performance and enhanced safety under impact. *Frontiers in Robotics and AI*, 2:34, 2015.

[183] Y. Kassahun and G. Sommer. Efficient reinforcement learning through Evolutionary Acquisition of Neural Topologies. In *ESANN*, pages 259–266, 2005.

[184] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

[185] M. Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In *European Conference on Genetic Programming*, pages 70–82. Springer, 2003.

[186] M. Keijzer. Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5(3):259–269, 2004.

[187] S. Kelly and M. I. Heywood. Emergent tangled graph representations for Atari game playing agents. In *Genetic Programming: 20th European Conference, EuroGP 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings 20*, pages 64–79. Springer, 2017.

[188] S. Kelly and M. I. Heywood. Multi-task learning in atari video games with emergent tangled program graphs. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 195–202, 2017.

[189] S. Kelly, T. Voegerl, W. Banzhaf, and C. Gondro. Evolving hierarchical memory-prediction machines in multi-task reinforcement learning. *Genetic Programming and Evolvable Machines*, 22:573–605, 2021.

[190] S. Kelly, D. S. Park, X. Song, M. McIntire, P. Nashikkar, R. Guha, W. Banzhaf, K. Deb, V. N. Boddeti, J. Tan, et al. Discovering Adaptable Symbolic Algorithms from Scratch. *arXiv preprint arXiv:2307.16890*, 2023.

[191] S. A. Kelly, T. M. Panhuis, and A. M. Stoehr. Phenotypic plasticity: molecular mechanisms and adaptive significance. *Comprehensive Physiology*, 2 (2):1417–1439, 2011.

[192] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[193] T. Klarner and E. P. Zehr. Sherlock Holmes and the curious case of the human locomotor central pattern generator. *Journal of neurophysiology*, 120(1):53–77, 2018.

[194] B. C. Kok and H. Soh. Trust in robots: Challenges and opportunities. *Current Robotics Reports*, 1(4):297–309, 2020.

[195] V. Kompella, R. Capobianco, S. Jong, J. Browne, S. Fox, L. Meyers, P. Wurman, and P. Stone. Reinforcement learning for optimization of COVID-19 mitigation policies. *arXiv preprint arXiv:2010.10560*, 2020.

[196] S. Koos, J.-B. Mouret, and S. Doncieux. Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 119–126, 2010.

[197] S. Koos, J.-B. Mouret, and S. Doncieux. The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, 17(1):122–145, 2012.

[198] O. Kosak, C. Wanninger, A. Hoffmann, H. Ponsar, and W. Reif. Multipotent systems: Combining planning, self-organization, and reconfiguration in modular robot ensembles. *Sensors*, 19(1):17, 2018.

[199] B. Kovalerchuk, M. A. Ahmad, and A. Teredesai. Survey of explainable machine learning with visual and granular methods beyond quasi-explanations. *Interpretable artificial intelligence: A perspective of granular computing*, pages 217–267, 2021.

[200] J. R. Koza. *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems*, volume 34. Stanford University, Department of Computer Science Stanford, CA, 1990.

[201] J. R. Koza. A genetic approach to finding a controller to back up a tractor-trailer truck. In *1992 American Control Conference*, pages 2307–2311. IEEE, 1992.

[202] J. R. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

[203] J. R. Koza and J. P. Rice. Automatic programming of robots using genetic programming. In *AAAI*, volume 92, pages 194–207, 1992.

[204] S. Kriegman, N. Cheney, and J. Bongard. How morphological development can guide evolution. *Scientific reports*, 8(1):13934, 2018.

[205] S. Kriegman, N. Cheney, F. Corucci, and J. C. Bongard. Interoceptive robustness through environment-mediated morphological development. *arXiv preprint arXiv:1804.02257*, 2018.

[206] S. Kriegman, S. Walker, D. Shah, M. Levin, R. Kramer-Bottiglio, and J. Bongard. Automated shapeshifting for function recovery in damaged robots. *arXiv preprint arXiv:1905.09264*, 2019.

[207] S. Kriegman, D. Blackiston, M. Levin, and J. Bongard. A scalable pipeline for designing reconfigurable organisms. *Proceedings of the National Academy of Sciences*, 117(4):1853–1859, 2020.

[208] S. Kriegman, A. M. Nasab, D. Shah, H. Steele, G. Branin, M. Levin, J. Bongard, and R. Kramer-Bottiglio. Scalable sim-to-real transfer of soft robot designs. In *2020 3rd IEEE International Conference on Soft Robotics (RoboSoft)*, pages 359–366. IEEE, 2020.

[209] L. Kunze, N. Hawes, T. Duckett, M. Hanheide, and T. Krajník. Artificial intelligence for long-term robot autonomy: A survey. *IEEE Robotics and Automation Letters*, 3(4):4023–4030, 2018.

[210] M.-K. Kvalsund, K. Glette, and F. Veenstra. Centralized and Decentralized Control in Modular Robots and Their Effect on Morphology. In *Conference on Artificial Life*, ALIFE '2023, page 49, 07 2023.

[211] W. La Cava, P. Orzechowski, B. Burlacu, F. O. de França, M. Virgolin, Y. Jin, M. Kommenda, and J. H. Moore. Contemporary symbolic regression methods and their relative performance. *arXiv preprint arXiv:2107.14351*, 2021.

[212] H. Lakkaraju, S. H. Bach, and J. Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1675–1684, 2016.

[213] M. Landajuela, B. K. Petersen, S. Kim, C. P. Santiago, R. Glatt, N. Mundhenk, J. F. Pettit, and D. Faissol. Discovering symbolic policies with deep reinforcement learning. In *International Conference on Machine Learning*, pages 5979–5989. PMLR, 2021.

[214] S. B. Laughlin, R. R. d. R. van Steveninck, and J. C. Anderson. The metabolic cost of neural information. *Nature neuroscience*, 1(1):36–41, 1998.

[215] N. Le. Organic selection and social heredity: The original baldwin effect revisited. In *Artificial Life Conference Proceedings*, pages 515–522. MIT Press, 2019.

[216] M. Lechner, R. Hasani, M. Zimmer, T. A. Henzinger, and R. Grosu. Designing worm-inspired neural networks for interpretable robotic control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 87–94. IEEE, 2019.

[217] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In *NIPs*, volume 2, pages 598–605. Citeseer, 1989.

[218] J. Legrand, S. Terryn, E. Roels, and B. Vanderborght. Reconfigurable, multi-material, voxel-based soft robots. *IEEE Robotics and Automation Letters*, 2023.

[219] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[220] J. Lehman and K. O. Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 211–218, 2011.

[221] A. Lensen. Mining Feature Relationships in Data. In *European Conference on Genetic Programming (Part of EvoStar)*, pages 247–262. Springer, 2021.

[222] A. Lensen, B. Xue, and M. Zhang. Genetic programming for evolving a front of interpretable models for data visualization. *IEEE transactions on cybernetics*, 51(11):5468–5482, 2020.

[223] J. Li, D. Li, R. Jiang, R. Xiao, H. Tang, and K. C. Tan. Vision-Action Semantic Associative Learning Based on Spiking Neural Networks for Cognitive Robot. *IEEE Computational Intelligence Magazine*, 17(4):27–38, 2022.

[224] W. Li, E. Buchanan Berumen, L. Le Goff, E. Hart, M. Hale, M. De Carlo, R. Woolley, A. Winfield, J. Timmis, A. Eiben, et al. Evaluation of frameworks that combine evolution and learning to design robots in complex morphological spaces. *IEEE Transactions on Evolutionary Computation*, 2023.

[225] X. Li and A. G.-O. Yeh. Neural-network-based cellular automata for simulating multiple land use changes using GIS. *International Journal of Geographical Information Science*, 16(4):323–343, 2002.

[226] Y. Li, Y. Zhong, J. Zhang, L. Xu, Q. Wang, H. Sun, H. Tong, X. Cheng, and X. Miao. Activity-dependent synaptic plasticity of a chalcogenide electronic synapse for neuromorphic systems. *Scientific reports*, 4(1):1–7, 2014.

[227] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022.

[228] X. Liang, X. Du, G. Wang, and Z. Han. A deep reinforcement learning network for traffic light cycle control. *IEEE Transactions on Vehicular Technology*, 68(2):1243–1253, 2019.

[229] X. Liao, A. V. Vasilakos, and Y. He. Small-world human brain networks: perspectives and challenges. *Neuroscience & Biobehavioral Reviews*, 77: 286–300, 2017.

[230] E. T. Liknes and D. L. Swanson. Phenotypic flexibility of body composition associated with seasonal acclimatization in passerine birds. *Journal of Thermal Biology*, 36(6):363–370, 2011.

[231] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi. Dynamic Model Pruning with Feedback. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SJem8lSFwB`.

[232] A. Lindenmayer. Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. *Journal of theoretical biology*, 18(3):280–299, 1968.

[233] H. Lipson. Challenges and opportunities for design, simulation, and fabrication of soft robots. *Soft Robotics*, 1(1):21–27, 2014.

[234] H. Lipson, V. Sunspiral, J. Bongard, and N. Cheney. On the difficulty of co-optimizing morphology and control in evolved virtual creatures. In *Artificial Life Conference Proceedings 13*, pages 226–233. MIT Press, 2016.

[235] Z. C. Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16 (3):31–57, 2018.

[236] D. Liu, M. Virgolin, T. Alderliesten, and P. A. Bosman. Evolvability degeneration in multi-objective genetic programming for symbolic regression. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 973–981, 2022.

[237] S. Liu, T. Chen, X. Chen, Z. Atashgahi, L. Yin, H. Kou, L. Shen, M. Pechenizkiy, Z. Wang, and D. C. Mocanu. Sparse Training via Boosting Pruning Plasticity with Neuroregeneration. *arXiv preprint arXiv:2106.10404*, 2021.

[238] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the Value of Network Pruning. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJlnB3C5Ym`.

[239] S. A. Lobov, A. N. Mikhaylov, M. Shamshin, V. A. Makarov, and V. B. Kazantsev. Spatial properties of STDP in a self-learning spiking neural network enable controlling a mobile robot. *Frontiers in neuroscience*, 14: 88, 2020.

[240] S. A. Lobov, A. I. Zharinov, V. A. Makarov, and V. B. Kazantsev. Spatial memory in a spiking neural network with robot embodiment. *Sensors*, 21 (8):2678, 2021.

[241] D. M. Lorenz, A. Jeng, and M. W. Deem. The emergence of modularity in biological systems. *Physics of life reviews*, 8(2):129–160, 2011.

[242] L. K. Low and H.-J. Cheng. Axon pruning: an essential step underlying the developmental plasticity of neuronal connections. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 361(1473):1531–1544, 2006.

[243] Y. Lu. Artificial intelligence: a survey on evolution, models, applications and future trends. *Journal of Management Analytics*, 6(1):1–29, 2019.

[244] Y. Lu, W. Wang, and L. Xue. A hybrid CNN-LSTM architecture for path planning of mobile robots in unknow environments. In *2020 Chinese Control And Decision Conference (CCDC)*, pages 4775–4779. IEEE, 2020.

[245] J. C. Lui and J. Baron. Mechanisms limiting body growth in mammals. *Endocrine reviews*, 32(3):422–440, 2011.

[246] S. Luke and L. Panait. A survey and comparison of tree generation algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 81–88. Citeseer, 2001.

[247] J. Luo, A. C. Stuurman, J. M. Tomczak, J. Ellers, and A. E. Eiben. The Effects of Learning in Morphologically Evolving Robot Systems. *Frontiers in Robotics and AI*, 9, 2022. ISSN 2296-9144. doi: 10.3389/frobt.2022.797393. URL https://www.frontiersin.org/articles/10.3389/frobt.2022.797393.

[248] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[249] A. Maedche, C. Legner, A. Benlian, B. Berger, H. Gimpel, T. Hess, O. Hinz, S. Morana, and M. Söllner. AI-based digital assistants: Opportunities, threats, and research perspectives. *Business & Information Systems Engineering*, 61:535–544, 2019.

[250] S. H. Mahdavi and P. J. Bentley. An evolutionary approach to damage recovery of robot motion with muscles. In *European Conference on Artificial Life*, pages 248–255. Springer, 2003. doi: doi.org/10.1007/b12035.

[251] Z. Mahoor, J. Felag, and J. Bongard. Morphology dictates a robot's ability to ground crowd-proposed language. *arXiv preprint arXiv:1712.05881*, 2017.

[252] A. Marrero, E. Segredo, E. Hart, J. Bossek, and A. Neumann. Generating diverse and discriminatory knapsack instances by searching for novelty in variable dimensions of feature-space. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 312–320, 2023.

[253] E. Masquil, G. Hamon, E. Nisioti, and C. Moulin-Frier. Intrinsically-Motivated Goal-Conditioned Reinforcement Learning in Multi-Agent Environments, 2022.

[254] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI magazine*, 27(4):12–12, 2006.

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

[256] E. Medvet and A. Bartoli. Evolutionary Optimization of Graphs with GraphEA. In *International Conference of the Italian Association for Artificial Intelligence*, pages 83–98. Springer, 2021.

[257] E. Medvet and G. Nadizar. GP for Continuous Control: Teacher or Learner? The Case of Simulated Modular Soft Robots. In *Genetic Programming Theory and Practice XX*, pages 203–224. Springer, 2024.

[258] E. Medvet and F. Rusin. Impact of Morphology Variations on Evolved Neural Controllers for Modular Robots. In *Italian Workshop on Artificial Life and Evolutionary Computation*, pages 266–277. Springer, 2022.

[259] E. Medvet, A. Bartoli, B. Carminati, and E. Ferrari. Evolutionary inference of attribute-based access control policies. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 351–365. Springer, 2015.

[260] E. Medvet, A. Bartoli, A. De Lorenzo, and G. Fidel. Evolution of distributed neural controllers for voxel-based soft robots. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 112–120, 2020.

[261] E. Medvet, A. Bartoli, A. De Lorenzo, and S. Seriani. 2D-VSR-Sim: A simulation tool for the optimization of 2-D voxel-based soft robots. *SoftwareX*, 12, 2020.

[262] E. Medvet, A. Bartoli, A. De Lorenzo, and S. Seriani. Design, Validation, and Case Studies of 2D-VSR-Sim, an Optimization-friendly Simulator of 2-D Voxel-based Soft Robots. *arXiv*, pages arXiv–2001, 2020.

[263] E. Medvet, A. Bartoli, F. Pigozzi, and M. Rochelli. Biodiversity in evolved voxel-based soft robots. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 129–137, 2021.

[264] E. Medvet, G. Nadizar, and F. Pigozzi. On the impact of body material properties on neuroevolution for embodied agents: the case of voxel-based soft robots. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 2122–2130, 2022.

[265] E. Medvet, G. Nadizar, F. Pigozzi, and E. Salvato. Evolutionary Machine Learning in Robotics. In *Handbook of Evolutionary Machine Learning*, pages 657–694. Springer, 2023.

[266] Y. Mei, Q. Chen, A. Lensen, B. Xue, and M. Zhang. Explainable artificial intelligence by genetic programming: A survey. *IEEE Transactions on Evolutionary Computation*, 27(3):621–641, 2022.

[267] A. Mertan and N. Cheney. Modular Controllers Facilitate the Co-Optimization of Morphology and Control in Soft Robots. In *Genetic and Evolutionary Computation Conference*, GECCO '23, page 174–183, New York, NY, USA, 2023. ACM. ISBN 9798400701191.

[268] A. Mertan and N. Cheney. Investigating Premature Convergence in Co-optimization of Morphology and Control in Evolved Virtual Soft Robots. In *European Conference on Genetic Programming (Part of EvoStar)*, pages 38–55. Springer, 2024.

[269] A. Mertan and N. Cheney. Towards Multi-Morphology Controllers with Diversity and Knowledge Distillation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 367–376, 2024.

[270] G. Methenitis, D. Hennes, D. Izzo, and A. Visser. Novelty search for soft robotic space exploration. In *Proceedings of the 2015 annual conference on Genetic and Evolutionary Computation*, pages 193–200, 2015.

[271] D. Meunier, R. Lambiotte, and E. T. Bullmore. Modular and hierarchically modular organization of brain networks. *Frontiers in neuroscience*, 4:200, 2010.

[272] T. Miconi, K. Stanley, and J. Clune. Differentiable plasticity: training plastic neural networks with backpropagation. In *International Conference on Machine Learning*, pages 3559–3568. PMLR, 2018.

[273] R. Miikkulainen, O. Francon, E. Meyerson, X. Qiu, D. Sargent, E. Canzani, and B. Hodjat. From prediction to prescription: evolutionary optimization of nonpharmaceutical interventions in the COVID-19 pandemic. *IEEE Transactions on Evolutionary Computation*, 25(2):386–401, 2021.

[274] T. Mildenberger. Stephen Marsland: Machine learning. An algorithmic perspective. *Statistical Papers*, 55(2):575, 2014.

[275] J. F. Miller. Cartesian genetic programming: its status and future. *Genetic Programming and Evolvable Machines*, 21:129–168, 2020.

[276] J. F. Miller and P. Thomson. Cartesian genetic programming. In *Genetic Programming*, pages 121–132, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-46239-2.

[277] K. Miras, E. Haasdijk, K. Glette, and A. Eiben. Effects of selection preferences on evolved robot morphologies and behaviors. In *Artificial Life Conference Proceedings*, pages 224–231. MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info . . . , 2018.

[278] K. Miras, E. Ferrante, and A. Eiben. Environmental influences on evolvable robots. *PloS one*, 15(5):e0233848, 2020.

[279] K. Miras, J. Cuijpers, B. Gülhan, and A. Eiben. The Impact of Early-death on Phenotypically Plastic Robots that Evolve in Changing Environments. In *ALIFE 2021: The 2021 Conference on Artificial Life*. MIT Press, 2021.

[280] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[281] A. Mohammadi, J. Lavranos, H. Zhou, R. Mutlu, G. Alici, Y. Tan, P. Choong, and D. Oetomo. A practical 3D-printed soft robotic prosthetic hand with multi-articulating capabilities. *PloS one*, 15(5):e0232766, 2020.

[282] C. Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[283] C. Molnar, G. König, J. Herbinger, T. Freiesleben, S. Dandl, C. A. Scholbeck, G. Casalicchio, M. Grosse-Wentrup, and B. Bischl. Pitfalls to avoid when interpreting machine learning models, 2020.

[284] F. Mondada, E. Franzi, and A. Guignard. The development of khepera. In *Experiments with the Mini-Robot Khepera, Proceedings of the First International Khepera Workshop*, pages 7–14, 1999.

[285] C. J. Moore, A. J. Chua, C. P. Berry, and J. R. Gair. Fast methods for training Gaussian processes on large datasets. *Royal Society open science*, 3(5):160125, 2016.

[286] A. Mordvintsev, E. Randazzo, E. Niklasson, and M. Levin. Growing neural cellular automata. *Distill*, 5(2):e23, 2020.

[287] A. Morel, Y. Kunimoto, A. Coninx, and S. Doncieux. Automatic acquisition of a repertoire of diverse grasping trajectories through behavior shaping and novelty search. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 755–761. IEEE, 2022.

[288] R. Moreno, F. Veenstra, D. Silvera, J. Franco, O. Gracia, E. Cordoba, J. Gomez, and A. Faina. Automated reconfiguration of modular robots using robot manipulators. In *2018 IEEE symposium series on computational intelligence (SSCI)*, pages 884–891. IEEE, 2018.

[289] A. S. Mori, T. Furukawa, and T. Sasaki. Response diversity determines the resilience of ecosystems to environmental change. *Biological reviews*, 88 (2):349–364, 2013.

[290] J.-B. Mouret and K. Chatzilygeroudis. 20 years of reality gap: a few thoughts about simulators in evolutionary robotics. In *Proceedings of the*

Genetic and Evolutionary Computation Conference Companion, pages 1121–1124, 2017.

[291] J.-B. Mouret and J. Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

[292] J.-B. Mouret and S. Doncieux. Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In *2009 IEEE congress on evolutionary computation*, pages 1161–1168. IEEE, 2009.

[293] J.-B. Mouret and S. Doncieux. Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evolutionary computation*, 20(1): 91–133, 2012.

[294] V. C. Müller and M. Hoffmann. What is morphological computation? On how the body contributes to cognition and control. *Artificial life*, 23(1): 1–24, 2017.

[295] A. Murphy, G. Murphy, J. Amaral, D. MotaDias, E. Naredo, and C. Ryan. Towards incorporating human knowledge in fuzzy pattern tree evolution. In *European Conference on Genetic Programming (Part of EvoStar)*, pages 66–81. Springer, 2021.

[296] G. Nadizar, E. Medvet, and K. Miras. On the Schedule for Morphological Development of Evolved Modular Soft Robots. In *European Conference on Genetic Programming (Part of EvoStar)*, pages 146–161. Springer, 2022.

[297] G. Nadizar, E. Medvet, S. Nichele, and S. Pontes-Filho. Collective control of modular soft robots via embodied Spiking Neural Cellular Automata. In *Workshop on From Cells to Societies: Collective Learning across Scales (Part of ICLR)*, 2022.

[298] G. Nadizar, E. Medvet, O. H. Ramstad, S. Nichele, F. A. Pellegrino, and M. Zullich. Merging pruning and neuroevolution: towards robust and efficient controllers for modular soft robots. *The Knowledge Engineering Review*, 37, 2022.

[299] G. Nadizar, E. Medvet, S. Nichele, and S. Pontes-Filho. An experimental comparison of evolved neural network models for controlling simulated modular soft robots. *Applied Soft Computing*, page 110610, 2023.

[300] G. Nadizar, E. Medvet, K. Walker, and S. Risi. A Fully-distributed Shape-aware Neural Controller for Modular Robots. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2023.

[301] G. Nadizar, E. Medvet, and D. Wilson. Searching for a Diversity of Interpretable Graph Control Policies. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 933–941, 2024.

[302] G. Nadizar, E. Medvet, and D. G. Wilson. Naturally interpretable control policies via graph-based genetic programming. In *European Conference on Genetic Programming (Part of EvoStar)*, pages 73–89. Springer, 2024.

[303] G. Nadizar, L. Rovito, A. De Lorenzo, E. Medvet, and M. Virgolin. An analysis of the ingredients for learning interpretable symbolic regression models with human-in-the-loop and genetic programming. *ACM Transactions on Evolutionary Learning*, 2024.

[304] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[305] E. Najarro and S. Risi. Meta-learning through hebbian plasticity in random networks. *arXiv preprint arXiv:2007.02686*, 2020.

[306] E. Najarro, S. Sudhakaran, and S. Risi. Towards Self-Assembling Artificial Neural Networks through Neural Developmental Programs. In *Conference on Artificial Life*, ALIFE '23, pages 1–10, 07 2023.

[307] M. Naumov, L. Chien, P. Vandermersch, and U. Kapasi. Cusparse library. In *GPU Technology Conference*, 2010.

[308] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson. Physically based deformable models in computer graphics. In *Computer graphics forum*, volume 25, pages 809–836. Wiley Online Library, 2006.

[309] B. Neyshabur, Z. Li, S. Bhojanapalli, Y. LeCun, and N. Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=BygfghAcYX`.

[310] S. Nichele, M. B. Ose, S. Risi, and G. Tufte. CA-NEAT: evolved compositional pattern producing networks for cellular automata morphogenesis and replication. *IEEE Transactions on Cognitive and Developmental Systems*, 10(3):687–700, 2017.

[311] J. Nielsen and H. H. Lund. Spiking neural building block robot with Hebbian learning. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 2, pages 1363–1369. IEEE, 2003.

[312] E. Niklasson, A. Mordvintsev, E. Randazzo, and M. Levin. Self-Organising Textures. *Distill*, 6(2):e00027–003, 2021.

[313] M. Nisser, L. Cheng, Y. Makaram, R. Suzuki, and S. Mueller. ElectroVoxel: Electromagnetically actuated pivoting for scalable modular self-reconfigurable robots. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 4254–4260. IEEE, 2022.

[314] G. Nitschke and D. Howard. AutoFac: The Perpetual Robot Machine. *IEEE Transactions on Artificial Intelligence*, 2021.

[315] S. Nolfi. *Behavioral and cognitive robotics: an adaptive perspective*. Stefano Nolfi, 2021.

[316] S. Nolfi and D. Floreano. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press, 2000.

[317] S. Nolfi and D. Floreano. Synthesis of autonomous robots through evolution. *Trends in cognitive sciences*, 6(1):31–37, 2002.

[318] S. Nolfi, J. Bongard, P. Husbands, and D. Floreano. *Evolutionary Robotics*, pages 2035–2068. Springer International Publishing, Cham, 2016. ISBN 978-3-319-32552-1. doi: 10.1007/978-3-319-32552-1_76. URL `https://doi.org/10.1007/978-3-319-32552-1_76`.

[319] J. Nordmoen, F. Veenstra, K. O. Ellefsen, and K. Glette. Quality and diversity in evolutionary modular robotics. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pages 2109–2116. IEEE, 2020.

[320] J. Nordmoen, F. Veenstra, K. O. Ellefsen, and K. Glette. Map-elites enables powerful stepping stones and diversity for modular robotics. *Frontiers in Robotics and AI*, 8:639173, 2021.

[321] C. D. Onal and D. Rus. A modular approach to soft robots. In *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pages 1038–1045. IEEE, 2012.

[322] M. O'Neill and C. Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001.

[323] R. OpenAI. Gpt-4 technical report. arxiv 2303.08774. *View in Article*, 2: 13, 2023.

[324] U. Orozco-Rosas, O. Montiel, and R. Sepúlveda. Mobile robot path planning using membrane evolutionary artificial potential field. *Applied Soft Computing*, 77:236–251, 2019.

[325] U. Orozco-Rosas, K. Picos, J. J. Pantrigo, A. S. Montemayor, and A. Cuesta-Infante. Mobile robot path planning using a QAPF learning algorithm for known and unknown environments. *IEEE Access*, 10:84648–84663, 2022.

[326] M. O'Neill. Riccardo Poli, William B. Langdon, Nicholas F. McPhee: a field guide to genetic programming, 2009.

[327] P. Pagliuca and S. Nolfi. The dynamic of body and brain co-evolution. *Adaptive Behavior*, 30(3):245–255, 2022.

[328] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[329] D. Pathak, C. Lu, T. Darrell, P. Isola, and A. A. Efros. Learning to control self-assembling morphologies: a study of generalization via modularity. *Advances in Neural Information Processing Systems*, 32, 2019.

[330] C. Paul. Morphological computation: A basis for the analysis of morphology and control requirements. *Robotics and Autonomous Systems*, 54(8): 619–630, 2006.

[331] J. W. Pedersen and S. Risi. Evolving and merging hebbian learning rules: increasing generalization by decreasing the number of rules. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 892–900, 2021.

[332] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

[333] A. G. Petzoldt and S. J. Sigrist. Synaptogenesis. *Current biology*, 24(22): R1076–R1080, 2014.

[334] R. Pfeifer and J. Bongard. *How the body shapes the way we think: a new view of intelligence*. MIT press, 2006.

[335] R. Pfeifer and F. Iida. Embodied artificial intelligence: Trends and challenges. *Lecture notes in computer science*, pages 1–26, 2004.

[336] C. Piazza, C. Della Santina, G. Grioli, A. Bicchi, and M. G. Catalano. Analytical Model and Experimental Testing of the SoftFoot: An Adaptive Robot Foot for Walking Over Obstacles and Irregular Terrains. *IEEE Transactions on Robotics*, 2024.

[337] F. Pigozzi, F. J. Camerota Verdù, and E. Medvet. How the Morphology Encoding Influences the Learning Ability in Body-Brain Co-Optimization. In *Genetic and Evolutionary Computation Conference*, pages 1045–1054, New York, NY, USA, 2023. ACM.

[338] F. Pigozzi, E. Medvet, A. Bartoli, and M. Rochelli. Factors Impacting Diversity and Effectiveness of Evolved Modular Robots. *ACM Transactions on Evolutionary Learning*, 3(1):1–33, 2023.

[339] R. Poli, W. B. Langdon, and N. F. McPhee. A Field Guide to Genetic Programming, 2008.

[340] S. Pontes-Filho and S. Nichele. A Conceptual Bio-Inspired Framework for the Evolution of Artificial General Intelligence. *arXiv preprint arXiv:1903.10410*, 2019.

[341] F. Poursabzi-Sangdeh, D. G. Goldstein, J. M. Hofman, J. W. Wortman Vaughan, and H. Wallach. Manipulating and measuring model interpretability. In *Proceedings of the 2021 CHI conference on human factors in computing systems*, pages 1–52, 2021.

[342] J. D. Power and B. L. Schlaggar. Neural plasticity across the lifespan. *Wiley Interdisciplinary Reviews: Developmental Biology*, 6(1):e216, 2017.

[343] S. G. R. Prabhu, R. C. Seals, P. J. Kyberd, and J. C. Wetherall. A survey on evolutionary-aided design in robotics. *Robotica*, 36(12):1804–1821, 2018.

[344] S. M. Prabhu and D. P. Garg. Artificial neural network based robot control: An overview. *Journal of Intelligent and Robotic Systems*, 15(4):333–365, 1996.

[345] S. Pratt, L. Weihs, and A. Farhadi. The Introspective Agent: Interdependence of Strategy, Physiology, and Sensing for Embodied Agents. *arXiv preprint arXiv:2201.00411*, 2022.

[346] A. F. Psaros, X. Meng, Z. Zou, L. Guo, and G. E. Karniadakis. Uncertainty quantification in scientific machine learning: Methods, metrics, and comparisons. *Journal of Computational Physics*, 477:111902, 2023.

[347] J. K. Pugh, L. B. Soros, and K. O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:202845, 2016.

[348] E. Puiutta and E. M. Veith. Explainable reinforcement learning: A survey. In *International cross-domain conference for machine learning and knowledge extraction*, pages 77–95. Springer, 2020.

[349] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[350] H. Qiu, M. Garratt, D. Howard, and S. Anavatti. Towards crossing the reality gap with evolved plastic neurocontrollers. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 130–138, 2020.

[351] D. V. Raman, A. P. Rotondo, and T. O'Leary. Fundamental bounds on learning performance in neural circuits. *Proceedings of the National Academy of Sciences*, 116(21):10537–10546, 2019.

[352] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092*, 2021.

[353] W. Ratajczak, P. Niedźwiedzka-Rystwej, B. Tokarz-Deptuła, and W. Deptuła. Immunological memory cells. *Central European Journal of Immunology*, 43(2):194–203, 2018.

[354] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911. PMLR, 2017.

[355] B. Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2: 253–279, 2019.

[356] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, B. B. Gupta, X. Chen, and X. Wang. A survey of deep active learning. *ACM computing surveys (CSUR)*, 54(9):1–40, 2021.

[357] A. Renda, J. Frankle, and M. Carbin. Comparing Fine-tuning and Rewinding in Neural Network Pruning. In *International Conference on Learning Representations*, 2020.

[358] J. Reuter, V. Martinek, R. Herzog, and S. Mostaghim. Unit-Aware Genetic Programming for the Development of Empirical Equations. In *International Conference on Parallel Problem Solving from Nature*, pages 168–183. Springer, 2024.

[359] M. T. Ribeiro, S. Singh, and C. Guestrin. " Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[360] M. M. Riccomagno and A. L. Kolodkin. Sculpting neural circuits by axon and dendrite pruning. *Annual review of cell and developmental biology*, 31: 779–805, 2015.

[361] S. Risi and K. O. Stanley. Indirectly encoding neural plasticity as a pattern of local rules. In *International Conference on Simulation of Adaptive Behavior*, pages 533–543. Springer, 2010.

[362] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[363] F. Rothlauf and D. E. Goldberg. Redundant representations in evolutionary computation. *Evolutionary Computation*, 11(4):381–415, 2003.

[364] L. Rovito, L. Bonin, L. Manzoni, and A. De Lorenzo. An Evolutionary Computation Approach for Twitter Bot Detection. *Applied Sciences*, 12 (12), 2022. ISSN 2076-3417. doi: 10.3390/app12125915. URL `https://www.mdpi.com/2076-3417/12/12/5915`.

[365] K. Roy and M. Foote. Morphological approaches to measuring biodiversity. *Trends in Ecology & Evolution*, 12(7):277–281, 1997.

[366] M. Rubenstein, A. Cornejo, and R. Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

[367] C. Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence*, 1(5):206–215, 2019.

[368] F. Rusin and E. Medvet. How Perception, Actuation, and Communication Impact the Emergence of Collective Intelligence in Simulated Modular Robots. *Artificial Life*, pages 1–18, 2024.

[369] O. Sagi and L. Rokach. Explainable decision forest: Transforming a decision forest into an interpretable tree. *Information Fusion*, 61:124–138, 2020. ISSN 1566-2535. doi: https://doi.org/10.1016/j.inffus.2020.

03.013. URL https://www.sciencedirect.com/science/article/pii/S1566253519307869.

[370] J. Sakai. Core Concept: How synaptic pruning shapes neural wiring during development and, possibly, in disease. *Proceedings of the National Academy of Sciences*, 117(28):16096–16099, 2020.

[371] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[372] E. Salvato, G. Fenu, E. Medvet, and F. A. Pellegrino. Crossing the Reality Gap: a Survey on Sim-to-Real Transferability of Robot Controllers in Reinforcement Learning. *IEEE Access*, 2021.

[373] W. Samek and K.-R. Müller. Towards explainable artificial intelligence. In *Explainable AI: interpreting, explaining and visualizing deep learning*, pages 5–22. Springer, 2019.

[374] E. Samuelsen and K. Glette. Some distance measures for morphological diversification in generative evolutionary robotics. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 721–728, 2014.

[375] E. Samuelsen and K. Glette. Real-world reproduction of evolved robot morphologies: Automated categorization and evaluation. In *European conference on the applications of evolutionary computation*, pages 771–782. Springer, 2015.

[376] F. Santosa and W. W. Symes. Linear inversion of band-limited reflection seismograms. *SIAM Journal on Scientific and Statistical Computing*, 7(4): 1307–1330, 1986.

[377] R. M. Sapolsky. *Behave: The biology of humans at our best and worst*. Penguin, 2017.

[378] O. Schuldiner and A. Yaron. Mechanisms of developmental neurite pruning. *Cellular and Molecular Life Sciences*, 72(1):101–119, 2015.

[379] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[380] J. Schulman, B. Zoph, C. Kim, J. Hilton, J. Menick, J. Weng, J. Uribe, L. Fedus, L. Metz, M. Pokorny, et al. ChatGPT: Optimizing language models for dialogue, 2022.

[381] E. Schwitzgebel. AI systems must not confuse users about their sentience or moral status. *Patterns*, 4(8), 2023.

[382] J. Secretan, N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, J. T. Folsom-Kovarik, and K. O. Stanley. Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evolutionary computation*, 19(3):373–403, 2011.

[383] B. Settles. Active learning literature survey, 2009.

[384] S. Sharma, J. Henderson, and J. Ghosh. Certifai: Counterfactual explanations for robustness, transparency, interpretability, and fairness of artificial intelligence models. *arXiv preprint arXiv:1905.07857*, 2019.

[385] C. A. Shaw, J. C. McEachern, and J. McEachern. *Toward a theory of neuroplasticity*. Psychology Press, 2001.

[386] R. K.-M. Sheh. " Why Did You Do That?" Explainable Intelligent Robots. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[387] Y. Shi, L. Nguyen, S. Oh, X. Liu, and D. Kuzum. A soft-pruning method applied during training of spiking neural networks for in-memory computing applications. *Frontiers in neuroscience*, 13:405, 2019.

[388] N. T. Siebel, J. Botel, and G. Sommer. Efficient neural network pruning during neuro-evolution. In *2009 International Joint Conference on Neural Networks*, pages 2920–2927. IEEE, 2009.

[389] O. Sigaud and F. Stulp. Policy search in continuous action domains: an overview. *Neural Networks*, 113:28–40, 2019.

[390] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[391] G. F. Smits and M. Kotanchek. Pareto-front exploitation in symbolic regression. In *Genetic programming theory and practice II*, pages 283–299. Springer, 2005.

[392] A. Spaeth, M. Tebyani, D. Haussler, and M. Teodorescu. Neuromorphic closed-loop control of a flexible modular robot by a simulated spiking central pattern generator. In *2020 3rd IEEE International Conference on Soft Robotics (RoboSoft)*, pages 46–51. IEEE, 2020.

[393] A. Spaeth, M. Tebyani, D. Haussler, and M. Teodorescu. Spiking neural state machine for gait frequency entrainment in a flexible modular robot. *PloS one*, 15(10):e0240267, 2020.

[394] C. Spearman. Footrule for measuring correlation. *British Journal of Psychology*, 2(1):89, 1906.

[395] O. Sporns. Structure and function of complex brain networks. *Dialogues in clinical neuroscience*, 15(3):247, 2013.

[396] O. Sporns, D. R. Chialvo, M. Kaiser, and C. C. Hilgetag. Complex networks: small-world and scale-free architectures. *Trends in Cognitive Sciences*, 9 (8):418–425, 2004.

[397] G. Squillero and A. Tonda. Divergence of character and premature convergence: A survey of methodologies for promoting diversity in evolutionary optimization. *Information Sciences*, 329:782–799, 2016.

[398] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[399] K. O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8 (2):131–162, 2007.

[400] K. O. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the 4th Annual Conference on genetic and evolutionary computation*, pages 569–577, 2002.

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

[402] L. Steels. When are robots intelligent autonomous agents? *Robotics and Autonomous systems*, 15(1-2):3–9, 1995.

[403] B. Strohmer, P. Manoonpong, and L. B. Larsen. Flexible spiking cpgs for online manipulation during hexapod walking. *Frontiers in neurorobotics*, 14:41, 2020.

[404] F. Stroppa, F. J. Majeed, J. Batiya, E. Baran, and M. Sarac. Optimizing soft robot design and tracking with and without evolutionary computation: an intensive survey. *Robotica*, pages 1–37, 2024.

[405] F. Stulp and O. Sigaud. Robot skill learning: From reinforcement learning to evolution strategies. *Paladyn, Journal of Behavioral Robotics*, 4(1):49–61, 2013.

[406] G. Su, D. Wei, K. R. Varshney, and D. M. Malioutov. Interpretable two-level boolean rule learning for classification. *arXiv preprint arXiv:1511.07361*, 2015.

[407] S. Sudhakaran, D. Grbic, S. Li, A. Katona, E. Najarro, C. Glanois, and S. Risi. Growing 3D Artefacts and Functional Machines with Neural Cellular Automata. *arXiv preprint arXiv:2103.08737*, 2021.

[408] X. Sui, H. Cai, D. Bie, Y. Zhang, J. Zhao, and Y. Zhu. Automatic generation of locomotion patterns for soft modular reconfigurable robots. *Applied Sciences*, 10(1):294, 2020.

[409] J. Talamini, E. Medvet, A. Bartoli, and A. De Lorenzo. Evolutionary Synthesis of Sensing Controllers for Voxel-based Soft Robots. In *Artificial Life Conference Proceedings*, pages 574–581. MIT Press, 2019.

[410] J. Talamini, E. Medvet, and S. Nichele. Criticality-Driven Evolution of Adaptable Morphologies of Voxel-Based Soft-Robots. *Frontiers in Robotics and AI*, 8:172, 2021.

[411] M. Tessler, M. R. Brugler, J. A. Burns, N. R. Sinatra, D. M. Vogt, A. Varma, M. Xiao, R. J. Wood, and D. F. Gruber. Ultra-gentle soft robotic fingers induce minimal transcriptomic response in a fragile marine animal. *Current Biology*, 30(4):R157–R158, 2020.

[412] G. Thimm and E. Fiesler. Evaluating pruning methods. In *Proceedings of the International Symposium on Artificial neural networks*, pages 20–25. Citeseer, 1995.

[413] H. H. Thodberg. Improving generalization of neural networks through pruning. *International Journal of Neural Systems*, 1(04):317–326, 1991.

[414] M. A. Thomis and B. Towne. Genetic determinants of prepubertal and pubertal growth and development. *Food and nutrition bulletin*, 27(4_suppl5): S257–S278, 2006.

[415] R. A. Thompson and C. A. Nelson. Developmental science and the media: Early brain development. *American Psychologist*, 56(1):5, 2001.

[416] S. L. Thomson, L. K. L. Goff, E. Hart, and E. Buchanan. Understanding fitness landscapes in morpho-evolution via local optima networks. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2024.

[417] T. G. Thuruthel, B. Shih, C. Laschi, and M. T. Tolley. Soft robot perception using embedded soft sensors and recurrent neural networks. *Science Robotics*, 4(26):eaav1488, 2019.

[418] Y. Tian. Artificial intelligence image recognition method based on convolutional neural network algorithm. *Ieee Access*, 8:125731–125744, 2020.

[419] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[420] R. Tibshirani. The lasso method for variable selection in the Cox model. *Statistics in medicine*, 16(4):385–395, 1997.

[421] D. Tilman, M. Clark, D. R. Williams, K. Kimmel, S. Polasky, and C. Packer. Future threats to biodiversity and pathways to their prevention. *Nature*, 546(7656):73–81, 2017.

[422] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[423] D. Trivedi, C. D. Rahn, W. M. Kier, and I. D. Walker. Soft robotics: Biological inspiration, state of the art, and future research. *Applied bionics and biomechanics*, 5(3):99–117, 2008.

[424] A. Trott, S. Srinivasa, D. van der Wal, S. Haneuse, and S. Zheng. Building a foundation for data-driven, interpretable, and robust policy design using the ai economist. *arXiv preprint arXiv:2108.02904*, 2021.

[425] A. Tsanas. *Accurate telemonitoring of Parkinson's disease symptom severity using nonlinear speech signal processing and statistical machine learning*. PhD thesis, Oxford University, UK, 2012.

[426] A. Tsanas and A. Xifara. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and buildings*, 49:560–567, 2012.

[427] G. G. Turrigiano and S. B. Nelson. Homeostatic plasticity in the developing nervous system. *Nature reviews neuroscience*, 5(2):97–107, 2004.

[428] R. J. Urbanowicz and J. H. Moore. Learning classifier systems: a complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications*, 2009, 2009.

[429] B. Ustun and C. Rudin. Supersparse linear integer models for optimized medical scoring systems. *Machine Learning*, 102(3):349–391, 2016.

[430] F. van Diggelen, E. Ferrante, N. Harrak, J. Luo, D. Zeeuwe, and A. Eiben. The Influence of Robot Traits and Evolutionary Dynamics on the Reality Gap. *IEEE Transactions on Cognitive and Developmental Systems*, 2021.

[431] L. Vanneschi, M. Castelli, and L. Manzoni. The k landscapes: a tunably difficult benchmark for genetic programming. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1467–1474, 2011.

[432] A. Variengien, S. Nichele, T. Glover, and S. Pontes-Filho. Towards self-organized control: Using neural cellular automata to robustly control a cart-pole agent. *arXiv preprint arXiv:2106.15240*, 2021.

[433] V. Vassiliades and J.-B. Mouret. Discovering the elite hypervolume by leveraging interspecies correlation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 149–156, 2018.

[434] V. Vassiliades, K. Chatzilygeroudis, and J.-B. Mouret. Scaling up map-elites using centroidal voronoi tessellations. *arXiv preprint arXiv:1610.05729*, 2016.

[435] V. Vassiliades, K. Chatzilygeroudis, and J.-B. Mouret. Using centroidal voronoi tessellations to scale up the multidimensional archive of phenotypic elites algorithm. *IEEE Transactions on Evolutionary Computation*, 22 (4):623–630, 2017.

[436] F. Veenstra, M. H. Olsen, and K. Glette. Effects of encodings and quality-diversity on evolving 2D virtual creatures. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 164–167, 2022.

[437] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pages 5045–5054. PMLR, 2018.

[438] B. Vézquez-Rodríguez, Z.-Q. Liu, P. Hagmann, and B. Misic. Signal propagation via cortical hierarchies. *Network Neuroscience*, 4(4):1072–1090, 2020.

[439] M. Videau, A. Leite, O. Teytaud, and M. Schoenauer. Multi-objective genetic programming for explainable reinforcement learning. In *European Conference on Genetic Programming (Part of EvoStar)*, pages 278–293. Springer, 2022.

[440] G. Vilone and L. Longo. Explainable artificial intelligence: a systematic review. *arXiv preprint arXiv:2006.00093*, 2020.

[441] M. Virgolin. genepro, 9 2022. URL `https://github.com/marcovirgolin/genepro`.

[442] M. Virgolin and S. P. Pissis. Symbolic Regression is NP-hard. *Transactions on Machine Learning Research*, 2022.

[443] M. Virgolin, T. Alderliesten, and P. A. Bosman. Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In *Proceedings of the genetic and evolutionary computation conference*, pages 1084–1092, 2019.

[444] M. Virgolin, T. Alderliesten, and P. A. Bosman. On explaining machine learning models by evolving crucial and compact features. *Swarm and Evolutionary Computation*, 53:100640, 2020.

[445] M. Virgolin, A. De Lorenzo, E. Medvet, and F. Randone. Learning a formula of interpretability to learn interpretable formulas. In *Parallel Problem Solving from Nature–PPSN XVI: 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part II 16*, pages 79–93. Springer, 2020.

[446] M. Virgolin, T. Alderliesten, C. Witteveen, and P. A. Bosman. Improving model-based genetic programming for symbolic regression of small expressions. *Evolutionary computation*, 29(2):211–237, 2021.

[447] M. Virgolin, A. De Lorenzo, F. Randone, E. Medvet, and M. Wahde. Model learning with personalized interpretability estimation (ML-PIE). In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1355–1364, 2021.

[448] E. K. Vladislavleva, G. Smits, and D. den Hertog. Order of Nonlinearity as a Complexity Measure for Models Generated by Symbolic Regression via Pareto Genetic Programming. *Evolutionary Computation, IEEE Transactions on*, 13:333 – 349, 05 2009. doi: 10.1109/TEVC.2008.926486.

[449] V. Vujovic, A. Rosendo, L. Brodbeck, and F. Iida. Evolutionary Developmental Robotics: Improving Morphology and Control of Physical Robots. *Artificial Life*, 23(2):169–185, 05 2017. ISSN 1064-5462.

[450] K. Walker, H. Hauser, and S. Risi. Growing simulated robots with environmental feedback: an eco-evo-devo approach. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 113–114, 2021.

[451] K. Walker, R. B. Palm, R. Moreno, A. Faina, K. Stoy, and S. Risi. Physical neural cellular automata for 2d shape classification. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12667–12673. IEEE, 2022.

[452] X. Wang, Z.-G. Hou, A. Zou, M. Tan, and L. Cheng. A behavior controller based on spiking neural networks for mobile robots. *Neurocomputing*, 71 (4-6):655–666, 2008.

[453] X. Wang, B. Yang, D. Tan, Q. Li, B. Song, Z.-S. Wu, A. del Campo, M. Kappl, Z. Wang, S. N. Gorb, et al. Bioinspired footed soft robot with unidirectional all-terrain mobility. *Materials Today*, 35:42–49, 2020.

[454] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[455] H. Wei, Y. Chen, J. Tan, and T. Wang. Sambot: A self-assembly modular robot system. *IEEE/ASME transactions on mechatronics*, 16(4):745–757, 2010.

[456] L. Wells and T. Bednarz. Explainable ai and reinforcement learning—a systematic review of current approaches and trends. *Frontiers in artificial intelligence*, 4:550030, 2021.

[457] J. Whitman, M. Travers, and H. Choset. Learning modular robot control policies. *IEEE Transactions on Robotics*, 2023.

[458] D. G. Wilson, S. Cussat-Blanc, H. Luga, and J. F. Miller. Evolving simple programs for playing Atari games. In *Proceedings of the genetic and evolutionary computation conference*, pages 229–236, 2018.

[459] D. G. Wilson, J. F. Miller, S. Cussat-Blanc, and H. Luga. Positional cartesian genetic programming. *arXiv preprint arXiv:1810.04119*, 2018.

[460] S. J. Woodman, D. S. Shah, M. Landesberg, A. Agrawala, and R. Kramer-Bottiglio. Stretchable arduinos embedded in soft robots. *Science Robotics*, 9(94):eadn6844, 2024.

[461] R. H. Wortham and A. Theodorou. Robot transparency, trust and utility. *Connection Science*, 29(3):242–248, 2017.

[462] F. Xu, H. Uszkoreit, Y. Du, W. Fan, D. Zhao, and J. Zhu. Explainable AI: A brief survey on history, research areas, approaches and challenges. In *CCF international conference on natural language processing and Chinese computing*, pages 563–574. Springer, 2019.

[463] A. Yaman, G. Iacca, D. C. Mocanu, M. Coler, G. Fletcher, and M. Pechenizkiy. Evolving plasticity for autonomous learning under changing environmental conditions. *Evolutionary computation*, 29(3):391–414, 2021.

[464] A. Yaman, J. Z. Leibo, G. Iacca, and S. W. Lee. The emergence of division of labor through decentralized social sanctioning, 2022.

[465] G. R. Yang, M. R. Joglekar, H. F. Song, W. T. Newsome, and X.-J. Wang. Task representations in neural networks trained to perform many cognitive tasks. *Nature neuroscience*, 22(2):297–306, 2019.

[466] Y. Yang and M. Loog. Active learning using uncertainty information. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2646–2651. IEEE, 2016.

[467] S. Ye, K. Xu, S. Liu, H. Cheng, J.-H. Lambrechts, H. Zhang, A. Zhou, K. Ma, Y. Wang, and X. Lin. Adversarial Robustness vs. Model Compression, or Both? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 111–120, 2019.

[468] H. You, C. Li, P. Xu, Y. Fu, Y. Wang, X. Chen, R. G. Baraniuk, Z. Wang, and Y. Lin. Drawing early-bird tickets: Towards more efficient training of deep networks. *arXiv preprint arXiv:1909.11957*, 2019.

[469] R. Yuste. From the neuron doctrine to neural networks. *Nature reviews neuroscience*, 16(8):487–497, 2015.

[470] A. Zador, S. Escola, B. Richards, B. Ölveczky, Y. Bengio, K. Boahen, M. Botvinick, D. Chklovskii, A. Churchland, C. Clopath, et al. Catalyzing next-generation artificial intelligence through neuroai. *Nature communications*, 14(1):1597, 2023.

[471] E. Zardini, D. Zappetti, D. Zambrano, G. Iacca, and D. Floreano. Seeking quality diversity in evolutionary co-design of morphology and control of soft tensegrity modular robots. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 189–197, 2021.

[472] M. N. Zennir, M. Benmohammed, and R. Boudjadja. Spike-Time Dependant Plasticity in a Spiking Neural Network for Robot Path Planning. In *AIAI Workshops*, pages 2–13, 2015.

[473] J. Zerilli, U. Bhatt, and A. Weller. How transparency modulates trust in artificial intelligence. *Patterns*, 3(4), 2022.

[474] B.-T. Zhang and H. Mühlenbein. Genetic programming of minimal neural nets using Occam's razor. In *Proceedings of the 5th international conference on genetic algorithms (ICGA'93*. Citeseer, 1993.

[475] C. Zhang, P. Zhu, Y. Lin, Z. Jiao, and J. Zou. Modular soft robotics: Modular units, connection mechanisms, and applications. *Advanced Intelligent Systems*, 2(6):1900166, 2020.

[476] Q. Zhang, P. Cui, D. Yan, J. Sun, Y. Duan, A. Zhang, and R. Xu. Whole-body humanoid robot locomotion with human reference. *arXiv preprint arXiv:2402.18294*, 2024.

[477] T. Zhang, X. Cheng, S. Jia, C. T. Li, M. ming Poo, and B. Xu. A brain-inspired algorithm that mitigates catastrophic forgetting of artificial and spiking neural networks with low computational cost. *Science Advances*, 9 (34):eadi2947, 2023.

[478] Z. Zhang. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–2. Ieee, 2018.

[479] Z. Zhang, D. Zhang, and R. C. Qiu. Deep reinforcement learning for power system applications: An overview. *CSEE Journal of Power and Energy Systems*, 6(1):213–225, 2019.

[480] R. Zhou and T. Hu. Evolutionary approaches to explainable machine learning. *arXiv preprint arXiv:2306.14786*, 2023.

[481] E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In *International conference on parallel problem solving from nature*, pages 832–842. Springer, 2004.

[482] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.

[483] X. Zou, E. Scott, A. Johnson, K. Chen, D. Nitz, K. De Jong, and J. Krichmar. Neuroevolution of a recurrent neural network for spatial and working memory in a simulated robotic environment. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 289–290, 2021.

[484] M. Zullich, E. Medvet, F. A. Pellegrino, and A. Ansuini. Speeding-up pruning for artificial neural networks: introducing accelerated iterative magnitude pruning. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 3868–3875. IEEE, 2021.

Curriculum Vitæ

Giorgia Nadizar was born in Trieste, Italy, on August 27th, 1997. She earned both her Bachelor's and Master's degrees in Electronics and Computer Engineering from the University of Trieste, graduating *cum laude* in 2019 and 2021, respectively.

During her Master's thesis in 2021, she spent six months working as a research assistant at the Oslo Metropolitan University, co-supervised by Prof. Eric Medvet and Prof. Stefano Nichele, where she developed a keen interest in bio-inspired Artificial Intelligence (AI).

In November 2021, Giorgia started her Ph.D. at the University of Trieste, under the supervision of Prof. Eric Medvet, in the Evolutionary Robotics and Artificial Life Lab. Her academic journey has allowed her to explore various research environments. At the end of 2022, she spent three months at the Centrum Wiskunde & Informatica (CWI) in Amsterdam, working with Dr. Marco Virgolin, where she discovered her passion for interpretability in AI. In 2023, she spent five months at ISAE-Supaero in Toulouse collaborating with Dr. Dennis Wilson. More recently, in 2024, she spent one month at the Massachusetts Institute of Technology (MIT) in the USA, hosted by the ALFA group led by Dr. Una-May O'Reilly, and another month in Toulouse at the University of Toulouse Capitole working with Prof. Sylvain Cussat-Blanc.

Throughout her Ph.D., Giorgia became increasingly involved in the research community focused on bio-inspired and evolutionary AI. She actively contributed to making the community more welcoming to students, serving as Student Affairs and Student Workshop Chair at the EvoStar Conference in both 2023 and 2024. She also advocated for inclusivity towards women and underrepresented minorities, co-organizing the Women+@GECCO initiative at the Genetic and Evolutionary Computation Conference in 2023.

In her efforts to promote interpretability in AI, Giorgia co-organized the Interpretable Control Competition and the Graph-based Genetic Programming Workshop at GECCO 2024. Additionally, she served as Proceedings Chair and Editor for the Conference on Artificial Life in 2024.

List of Publications

27. A. Ferigo*, G. Iacca*, E. Medvet*, and **G. Nadizar*** (* equal contribution). Totipotent Neural Controllers for Modular Soft Robots: Achieving Specialization in Body-Brain Co-Evolution through Hebbian Learning. *Neurocomputing*. 2024

26. M. El Saliby, E. Medvet, **G. Nadizar**, E. Salvato, and S. L. Thomson. Factors Impacting Landscape Ruggedness in Control Problems. In *XVIII International Workshop on Artificial Life and Evolutionary Computation (WIVACE)*, 2024

25. **G. Nadizar**, B. Sakallioglu, F. Garrow, S. Silva, and L. Vanneschi. Geometric semantic GP with linear scaling: Darwinian versus Lamarckian evolution. *Genetic Programming and Evolvable Machines*, 25(2):1–24, 2024

24. **G. Nadizar**, L. Rovito, D. G. Wilson, and E. Medvet. Interpretable Control Competition. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 11–12, 2024

23. M. El Saliby, **G. Nadizar**, E. Salvato, and E. Medvet. Eventually, all you need is a simple evolutionary algorithm (for neuroevolution of continuous control policies). In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1904–1913, 2024

22. S. Jorgensen*, **G. Nadizar***, G. Pietropolli*, L. Manzoni, E. Medvet, U.-M. O'Reilly, and E. Hemberg (* shared first co-author). Large Language Model-based Test Case Generation for GP Agents. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 914–923, 2024
☆ *Nomination for best paper award*

21. **G. Nadizar**, E. Medvet, and D. Wilson. Searching for a Diversity of Interpretable Graph Control Policies. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 933–941, 2024

20. **G. Nadizar** and E. Medvet. Interpretable Control of Modular Soft Robots. In *French Regional Conference on Complex Systems (FRCCS)*, 2024

19. **G. Nadizar**⋆, L. Rovito⋆, A. De Lorenzo, E. Medvet, and M. Virgolin (⋆ shared first co-author). An analysis of the ingredients for learning interpretable symbolic regression models with human-in-the-loop and genetic programming. *ACM Transactions on Evolutionary Learning*, 2024

18. **G. Nadizar**, E. Medvet, and D. G. Wilson. Naturally interpretable control policies via graph-based genetic programming. In *European Conference on Genetic Programming (Part of EvoStar)*, pages 73–89. Springer, 2024
★ *Best paper award*

17. G. Marchiori Pietrosanti, **G. Nadizar**, F. Pigozzi, and E. Medvet. Human Control of Simulated Modular Soft Robots may Predict the Performance of Optimized AI-based Controllers. *IEEE Access*, 2023

16. E. Medvet, **G. Nadizar**, F. Pigozzi, and E. Salvato. Evolutionary Machine Learning in Robotics. In *Handbook of Evolutionary Machine Learning*, pages 657–694. Springer, 2023

15. **G. Nadizar**⋆ and G. Pietropolli⋆ (⋆ equal contribution). A grammatical evolution approach to the automatic inference of P systems. *Journal of Membrane Computing*, 2023

14. **G. Nadizar**, E. Medvet, S. Nichele, and S. Pontes-Filho. An experimental comparison of evolved neural network models for controlling simulated modular soft robots. *Applied Soft Computing*, page 110610, 2023

13. **G. Nadizar**, E. Medvet, K. Walker, and S. Risi. Neural Cellular Automata Enable Self-Discovery of Physical Configuration in Modular Robots Driven by Collective Intelligence. In *Proceedings of the Distributed Ghost Workshop of the Artificial Life Conference*, pages 1–3. Stefano Nichele, 2023

12. E. Medvet and **G. Nadizar**. GP for Continuous Control: Teacher or Learner? The Case of Simulated Modular Soft Robots. In *Genetic Programming Theory and Practice XX*, pages 203–224. Springer, 2024

11. **G. Nadizar**, E. Medvet, K. Walker, and S. Risi. A Fully-distributed Shape-aware Neural Controller for Modular Robots. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2023

10. **G. Nadizar**, F. Garrow, B. Sakallioglu, L. Canonne, S. Silva, and L. Vanneschi. An Investigation of Geometric Semantic GP with Linear Scaling. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1165–1174, 2023

9. **G. Nadizar** and E. Medvet. On the Effects of Collaborators Selection and Aggregation in Cooperative Coevolution: An Experimental Analysis. In *Genetic Programming: 26th European Conference, EuroGP 2023, Held as Part of EvoStar 2023, Brno, Czech Republic, April 12–14, 2023, Proceedings*, pages 292–307. Springer, 2023

8. **G. Nadizar*** and G. Pietropolli* (* equal contribution). P Systems inference via Grammatical Evolution. In *23rd Conference on Membrane Computing (CMC)*, 2022

7. S. Furlan, E. Medvet, **G. Nadizar**, and F. Pigozzi. On the Mutual Influence of Human and Artificial Life: an Experimental Investigation. In *ALIFE 2022: The 2022 Conference on Artificial Life*. MIT press, 2022

6. E. Medvet, **G. Nadizar**, and F. Pigozzi. On the impact of body material properties on neuroevolution for embodied agents: the case of voxel-based soft robots. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 2122–2130, 2022

5. E. Medvet, **G. Nadizar**, and L. Manzoni. JGEA: a modular java framework for experimenting with evolutionary computation. In *Proceedings of the genetic and evolutionary computation conference companion*, pages 2009–2018, 2022

4. **G. Nadizar**, E. Medvet, S. Nichele, and S. Pontes-Filho. Collective control of modular soft robots via embodied Spiking Neural Cellular Automata. In *Workshop on From Cells to Societies: Collective Learning across Scales (Part of ICLR)*, 2022
   ★ *Best poster award*

3. **G. Nadizar**, E. Medvet, and K. Miras. On the Schedule for Morphological Development of Evolved Modular Soft Robots. In *European Conference on Genetic Programming (Part of EvoStar)*, pages 146–161. Springer, 2022
   ★ *Best student poster award*

2. **G. Nadizar**, E. Medvet, O. H. Ramstad, S. Nichele, F. A. Pellegrino, and M. Zullich. Merging pruning and neuroevolution: towards robust and efficient controllers for modular soft robots. *The Knowledge Engineering Review*, 37, 2022

1. **G. Nadizar**, E. Medvet, F. A. Pellegrino, M. Zullich, and S. Nichele. On the effects of pruning on evolved neural controllers for soft robots. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1744–1752, 2021.

Acknowledgments

"Bravi tutti" is a common saying in our lab to acknowledge a collective effort. It symbolizes the importance of everyone in the accomplishment of some goal. When I look back at my Ph.D. journey, I can't avoid thinking "bravi tutti". These three years have been an enriching and stimulating adventure, and I am full of gratitude for all the people who have guided me and walked beside me through it all.

First, my heartfelt thank you goes to my supervisor prof. Eric Medvet, who guided and supported me through my entire Ph.D. Thank you for teaching me academic rigor, the importance of precision, and attention to detail. Thank you for being a model of hard work, but above all healthy work, with breaks, laughter, and good relationships among collaborators. Thank you for guiding me when I needed direction, but also for leaving me the academic freedom to pursue my own ideas. I would also like to express my gratitude to Stefano Nichele, my co-supervisor, who never failed to advise me when I needed it, even from far away.

I wish to further thank all the people who hosted me during my academic visits around the globe. Thank you for your hospitality, and for contributing to making me a better and more independent researcher and person. I would like to thank Marco Virgolin, Peter Bosman, and the entire team at CWI, Una-May O'Reilly, Erik Hemberg, and the ALFA group at MIT, Sylvain Cussat-Blanc and the REVA team at UT Capitole. A special thank you goes to Dennis G. Wilson, who instilled in me the passion for graph-based genetic programming, one of my main areas of research currently. Thank you Dennis for hosting me and inviting me countless times to Toulouse at ISAE-SUPAERO, for giving me selfless advice, and for our stimulating research discussions.

I am also thankful to my colleagues at the ERALLAB and my extended research team in Trieste for all the discussions and the time spent together. Thank you for being there when I needed you and thank you for never failing to make me laugh in all coffee and lunch breaks. Thank you for creating a work environment I look forward to going to every day. From the Trieste team, I wish to specifically thank Erica, who helped me navigate the academic world since the beginning, sharing her experience and being like a big sister to me. I would also like to give special thanks to Gloria, with whom I shared most of my academic

adventures, from the USA to Australia. Thank you for always being there and for being a great friend, before being a good colleague.

I would also like to thank all the researchers who I had the fortune to co-author papers with: each and every one of you has enriched me as a researcher and taught me something. Looking back at these three years I feel particularly grateful for the network I have built.

I also wish to express my gratitude to the entire Evolutionary Computation community. Thank you to all the people who made me feel welcome from day 0 and to all those who worked hard to make it a healthy and inclusive environment. Among these, I wish to particularly acknowledge the SPECIES society, which funded one of my research periods abroad and actively works to make the community better for all. I am grateful to be part of this.

Finally, on a more personal note, I wish to thank all the people who stood by my side along this path, celebrating even my smallest successes and cheering me up when I needed it. Thanks to my family and friends who were there all along, I know I never thank you enough, but I am lucky to have you. Thank you for always believing in me and for helping me believe in myself.