AI

MDPI

Article

Learning and Evolution: Factors Influencing an Effective Combination

Paolo Pagliuca

Institute of Cognitive Sciences and Technologies (ISTC), National Research Council (CNR), Via Giandomenico Romagnosi 18A, 00196 Rome, Italy; paolo.pagliuca@istc.cnr.it

Abstract: (1) Background: The mutual relationship between evolution and learning is a controversial argument among the artificial intelligence and neuro-evolution communities. After more than three decades, there is still no common agreement on the matter. (2) Methods: In this paper, the author investigates whether combining learning and evolution permits finding better solutions than those discovered by evolution alone. In further detail, the author presents a series of empirical studies that highlight some specific conditions determining the success of such combination. Results are obtained in five qualitatively different domains: (i) the 5-bit parity task, (ii) the double-pole balancing problem, (iii) the Rastrigin, Rosenbrock and Sphere optimization functions, (iv) a robot foraging task and (v) a social foraging problem. Moreover, the first three tasks represent benchmark problems in the field of evolutionary computation. (3) Results and Discussion: The outcomes indicate that the effect of learning on evolution depends on the nature of the problem. Specifically, when the problem implies limited or absent agent–environment conditions, learning is beneficial for evolution, especially with the introduction of noise during the learning and selection processes. Conversely, when agents are embodied and actively interact with the environment, learning does not provide advantages, and the addition of noise is detrimental. Finally, the absence of stochasticity in the experienced conditions is paramount for the effectiveness of the combination. Furthermore, the length of the learning process must be fine-tuned based on the considered task.

Keywords: evolution; learning; memetic algorithms; stochastic hill-climbing; evolutionary strategies

Citation: Pagliuca, P. Learning and Evolution: Factors Influencing an Effective Combination. *AI* **2024**, *5*, 2393–2432. https://doi.org/10.3390/ai5040118

Received: 3 September 2024
Revised: 25 October 2024
Accepted: 12 November 2024
Published: 15 November 2024

1. Introduction

The interplay between learning and evolution has been studied for decades, but it is still a very controversial topic. Despite the huge amount of work, to what extent the interaction between learning and evolution actually fosters the development of successful behaviors is still a matter of debate in the scientific community. Indeed, as reported in [1,2], there exist some controversial arguments about the effect of learning on evolution. Some studies revealed how learning accelerates evolution [3–18], while other works demonstrated that learning does not provide any advantage on the course of evolution [19–25].

As explained in [26], "Evolution and learning (or phylogenetic and ontogenetic adaptation) are two forms of biological adaptation that differ in space and time. Evolution is a process of selective reproduction and substitution based on the existence of a population of individuals displaying variability at the genetic level. Learning, instead, is a set of modifications taking place within each single individual during its own lifetime. Evolution and learning operate on different time scales. Evolution is a form of adaptation capable of capturing relatively slow environmental changes that might encompass several generations (e.g., the perceptual characteristics of food sources for a given species). Learning, instead, allows an individual to adapt to environmental modifications that are unpredictable at the generational level. Learning might include a variety of mechanisms that produce adaptive changes in an individual during its lifetime, such as physical development, neural maturation, variation of the connectivity between neurons, and synaptic plasticity. Finally,

whereas evolution operates on the genotype, learning affects only the phenotype and phenotypic modifications cannot directly modify the genotype". Therefore, learning and evolution provide two alternative frameworks [27] to understand the adaptive changes allowing evolving agents to behave more effectively based on the particular environment they are situated in. However, learning and evolution might concur in the development of complex behaviors. The first work trying to highlight the positive effect of learning on evolution is presented in [8]. The authors considered a simple experimental setting, where genotype and phenotype representations are trivial and their relationship is immediate. The genotype is a string of bits, while the phenotype is a neural network. Given a genotype, a 1-bit corresponds to the presence of a particular connection in the network. Conversely, a 0-bit means that the corresponding connection is absent. In the abstract task used by the authors, only a specific combination of genes (i.e., a genotype with all 1-bits) obtains a fitness score of 1, whereas all other genotypes receive a fitness score of 0. To study the effect of the combination of evolution and learning, the authors considered a control situation in which the alleles could also assume a "*" value and learning operated by simply assigning random values to these alleles. The collected results demonstrated how the combination of learning and evolution permits finding the solution of the problem, while the use of learning or evolution alone fails. To date, however, this method has not been successfully applied to realistic problems, such as, for example, the evolution of robots selected based on their capability to solve a problem that cannot be addressed by using evolution alone.

The seminal work [8] has been a source of inspiration for many other works. Some studies stated that the combination of evolution and learning is advantageous compared to the application of single approaches [3,6,7,9,16,17,28–30], while others showed that learning actually decelerates evolution [19–22,24,25,31]. Moreover, some works focused on the analysis of the benefits and limitations of plasticity/learning or the conditions under which learning accelerates/decelerates evolution [32–40]. In spite of the huge amount of publications arguing that learning has a beneficial/detrimental effect on evolution, there are aspects that have not yet been considered and require a deeper analysis. In particular, two weak points can be found in most of the cited studies. First, the majority of these works do not consider the computational costs, a crucial factor in determining the superiority/inferiority of a method over another. Evaluating different algorithms under the same evolutionary conditions and for the same duration is pivotal to shed light on and untangle this topic. Second, most of the conducted studies have been carried out by using trivial and often abstract problems specifically designed to test the proposed algorithms. Instead, the main goal of this work is to verify whether the combination of learning and evolution might be advantageous over traditional approaches in widespread and more challenging domains.

In this work, the author introduces a new technique, called Stochastic Steady State with Hill-Climbing (SSSHC), which combines an Evolutionary Algorithm (EA), the Stochastic Steady State (SSS) [41], with an optimization method performing solution modifications. The SSSHC algorithm belongs to the family of Memetic Algorithms (MAs) [42,43]. MAs operate at a population level like traditional EAs, but they allow the refinement of solutions through a local search technique retaining adaptive modifications. This enables to improve the quality of the solutions found and helps to avoid the premature convergence issue. In [44], the authors demonstrate the superiority of MAs over EAs in the hurdle problems [45]. A memetic algorithm combining a classic genetic algorithm (GA) [46] and hill-climbing (HC) [47] as a refinement strategy has been proposed in [48]. It is worth highlighting that the term "learning" here denotes a refinement process consisting of repeatedly generating modifications of current solutions, verifying whether such changes produce an advantage and retaining positive modifications. Specifically, the learning process used in the presented experiments is obtained through a stochastic hill-climbing process [49], exploring the search space and looking for adaptive solutions. Variations concurring with such adaptive traits are inherited in the population. The reader can imagine this process as a form of Lamarckian learning [50,51]. Lamarckian learning has been successfully used

for solving function optimization problems [17], evolving solutions in multi-agent environments [51], training recurrent neural networks [52,53], training convolutional neural networks for image classification [54] and evolving robot body and brain [55–59]. A comparison of the performance of GA and MA using both Baldwinian and Lamarckian learning is reported in [60]. In addition, in [61] the authors show how the combination of evolution and Lamarckian learning enables the discovery of more effective solutions than the single approaches in two robotic scenarios. The presented examples indicate that, under particular circumstances, learning promotes the identification of behaviors superior to those provided through evolution.

The aim of this paper is to investigate whether the combination of learning and evolution permit finding better solutions than evolution alone and what factors foster the success or failure of such a combination. This has been examined in five different domains: (i) the well-known 5 bit-parity task, (ii) the popular double-pole balancing problem [62], a benchmark task largely used to compare different evolutionary algorithms [41,63–66], (iii) the Rastrigin [67], Rosenbrock [68] and Sphere optimization functions, which constitute benchmark problems in evolutionary computation and optimization [69,70], (iv) a robot foraging task and (v) a social foraging task already used to evaluate dynamics, adaptation and emergent behaviors in two-agent systems [71–73]. The collected results indicate that the combination of evolution and learning allows discovering better solutions than evolution alone when (i) the task implies absent or limited agent–environment interactions and (ii) the problem is deterministic, i.e., the experienced conditions are kept constant throughout the whole evolutionary process. Under these circumstances, the addition of noise to the learning process is advantageous and the computational cost required by the combination of learning and evolution to find out a suitable solution is lower. Conversely, learning does not improve the quality of solutions discovered by evolution when the considered problem involves embodied agents interacting with the environment, and the addition of noise has a detrimental effect. Finally, the addition of learning to evolution does not provide advantages in the function optimization scenario. A noteworthy aspect concerns the fact that the comparison has been performed by using the same parameter settings in order to avoid biases in the results. In particular, different combinations of parameters have been systematically analyzed and the outcomes are provided in the Appendix Appendix A. The author did not use automatic algorithm configuration methods [74,75], since the full list of parameters to be configured requires significantly more powerful computational resources and goes beyond the scope of this research. In fact, not only the algorithm's parameters but also the controller's parameters have to be optimized. Moreover, the considered problems have already been addressed in the literature [41,63–66,70,72,76,77], and some of the parameters are derived from these works.

The main contributions of this work can be summarized as follows:

- an analysis of the factors promoting a successful combination of evolution and learning is provided;
- the conditions under which learning is not beneficial to evolution are investigated and discussed;
- a novel evolutionary algorithm, called Stochastic Steady State with Hill-Climbing (SSSHC), is proposed and compared with the two state-of-the art methods constituting it in five different experimental scenarios;
- the combination of learning and evolution is beneficial when there are limited or absent agent–environment interactions and the experienced conditions do not change during evolution. Under these circumstances, the addition of noise to both the selection and the learning processes provides advantages to evolution;
- learning does not improve the solutions found by evolution when the considered problem involves embodied agents actively interacting with the environment. Moreover, in these scenarios, the addition of learning has a detrimental effect;
- learning is not effective at increasing the performance of solutions discovered by evolution in the function optimization setting.

In the next section, the author presents the methodology used to investigate the factors fostering the emergence of a beneficial effect of learning on evolution. Specifically, in Section 2.1, the evolutionary tasks used in this work are illustrated, while Section 2.2 contains a description of the different evolutionary algorithms used. In Section 3, the results of the performed analysis are presented. Section 4 provides a discussion of the experimental outcomes and identifies the main findings of the work. Finally, in Section 5, the author draws his conclusions.

2. Materials and Methods

This section provides a thorough explanation of the methodology used to validate the research hypothesis. In particular, the evolutionary tasks are described in Section 2.1, while the algorithms are presented in Section 2.2.

2.1. Tasks

Three of the five considered tasks represent well-known benchmark problems in evolutionary computation, while the other two problems involve the use of robotic agents interacting with and modifying the environment in which they are situated. As stated in Section 1, the main purpose is to verify to what extent learning is beneficial to evolution in widely recognized and challenging domains.

The experiments were run using the Framework for Autonomous Robotics Simulation and Analysis (FARSA) simulator [78,79], an open software tool widely used to implement simulations for autonomous robot controllers [61,71–73,80–83].

2.1.1. 5-Bit Parity

Digital circuits (Figure 1) are systems computing logic functions, like the sum and/or the multiplication of digital numbers. The input of these systems consists of two or more binary (Boolean) values, while the output includes one or more binary values. A digital circuit is made of several logic gates receiving two binary values in input and producing one binary value in output, which is the result of an elementary logic function (e.g., AND, OR, NAND, NOR) of the input. The input of each gate can come from either the input pattern or the output of other gates. The types of wiring between gates and the logic functions computed by each gate determine the logic function calculated by the circuit [76].

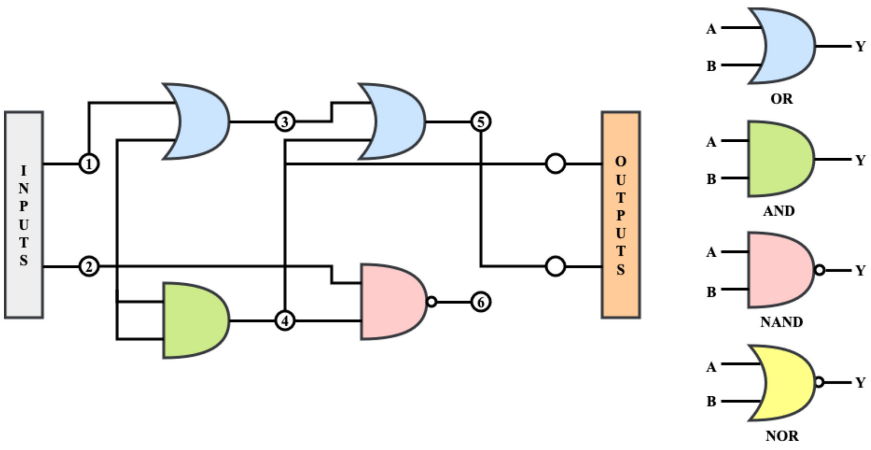

Figure 1. An example of a digital circuit: it consists of four logic gates, receives two inputs and produces two outputs. On the right side, there are four symbols corresponding to the four types of usable logic gates. The numbers 1–2 indicate the binary states constituting the inputs of the circuit. The numbers 3–6 represent the outputs calculated by the associated gates. The circuit output is given by the outputs of the two logic gates wired to the output units (in this example, the output is given by gates 4 and 5). The lines indicate how gates are wired. Figure adapted from [76].

Digital circuits can be either hardwired or simulated. In this work, the second scenario was considered, in which circuits evolve: they are represented through genotypes encoding the way gates are wired and logic function calculated by each gate. The ability of the evolving circuits to approximate a certain function is evaluated by computing a fitness score measuring the error between the circuit outputs and the corresponding target values.

In the experiments reported in this paper, the task is to evolve simulated digital circuits for the ability to calculate a 5-bit even parity function (i.e., the capability of producing an output equal to 1 when the number of 1-bits is even). The architecture of the evolving circuits was derived from [76]: they are made of 400 logic gates divided into 20 layers of 20 gates. The evolution of circuits implementing a 5-bit parity function by using only OR, AND, NAND and NOR logic gates represents a far from trivial problem [76,77,84].

The evaluation of circuits consists of measuring their ability to produce the desired outputs for all the 2^n (32 in this case, with $n = 5$) possible input patterns. The fitness is computed according to Equation (1):

$$F = 1 - \frac{1}{2^n} \sum_{j=1}^{2^n} |O_j - E_j| \quad (1)$$

In Equation (1), n denotes the size of the input pattern of the circuit, j indicates the generic input pattern (with $j \in [1, 2^n]$), O_j represents the output produced by the circuit when receiving the pattern j, and E_j is the expected output for pattern j.

A circuit evaluation requires one evolutionary step. The evolutionary process is continued until the total number of performed steps reaches 10^8, i.e., when around $\frac{10^8}{32} = 3.125 \times 10^6$ circuits have been evaluated.

2.1.2. Double-Pole Balancing

The double-pole balancing problem, introduced in [62], is a benchmark task in which two poles are attached to a mobile cart through passive hinge joints. The experimental settings are provided in Table 1. The goal is to move the cart within the track in order to prevent the poles from falling. An illustration of the problem is shown in Figure 2.

Table 1. Double-pole balancing parameters. All the values are expressed with their units of measurement. The ratio between the size of the long pole and that of the short one is 10 to 1. The cart moves only along one dimension x (see Figure 2).

Parameter	Value
cart mass	1 kg
long pole length	1.0 m
long pole mass	0.5 kg
short pole length	0.1 m
short pole mass	0.05 kg
track length	4.8 m

In this work, the author analyzed only the non-Markovian version of the task, i.e., the case in which no velocity input is provided and the system has to derive this information internally. To this end, the agent's controller is a recurrent neural network receiving four sensory inputs (see Table 2).

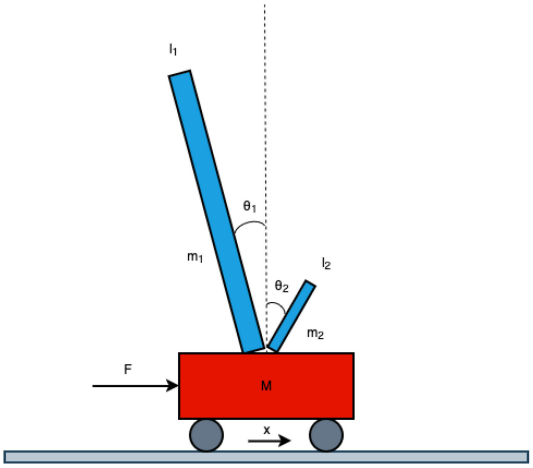

Figure 2. The double-pole balancing problem. Two poles (blue rectangles) are placed on a wheeled cart (red rectangle, the wheels are represented by two dark gray circles), which can move only on the x-axis in a limited track (light gray rectangle over which the cart is placed). The goal is to avoid both poles falling and the cart exiting from the track.

Table 2. List of inputs of the non-Markovian double-pole balancing problem with their associated ranges. The "-" symbol denotes the absence of the corresponding entry.

Input	Description	Range
x	Position of the cart on the track	$[-2.4, 2.4]$ m
θ_1	Angle of the long pole	$[-36°, 36°]$
θ_2	Angle of the short pole	$[-36°, 36°]$
$b = 0.5$	Constant bias	-

The network output determines the force applied to the cart and is normalized in the range $[-10.0, 10.0]$ N. The states of the x, θ_1 and θ_2 sensors are normalized in the $[-0.5, 0.5]$, $\left[-\frac{5}{13} \times \pi, \frac{5}{13} \times \pi\right]$ and $\left[-\frac{5}{13} \times \pi, \frac{5}{13} \times \pi\right]$ ranges, respectively [41]. Equations (2)–(9) [62] are used to compute the effective mass of the poles (Equation (2)), the acceleration of the poles (Equation (3)), the acceleration of the cart (Equation (4)), the effective force on each pole (Equation (5)), the next angle of the poles (Equation (6)), the velocity of the poles (Equation (7)), the position of the cart (Equation (8)), and the velocity of the cart (Equation (9)), respectively.

$$\hat{m}_i = m_i \times \left(1 - \frac{3}{4} cos^2(\theta_i)\right) \tag{2}$$

$$\dot{\theta}_i = -\frac{3}{4 * l_i}\left(\ddot{x} \times cos(\theta_i) + g \times sin(\theta_i) + \frac{\mu_{pi} \times \dot{\theta}_i}{m_i \times l_i}\right) \tag{3}$$

$$\ddot{x} = \frac{F + \sum_{i=0}^{N} \hat{F}_i}{M + \sum_{i=0}^{N} \hat{M}_i} \tag{4}$$

$$\begin{aligned} \hat{F}_i = & \; m_i \times l_i \times \dot{\theta}_i^{\,2} \times sin(\theta_i) \\ & + \frac{3}{4} \times m_i \times cos(\theta_i) \times \left(\frac{\mu_{pi} \times \dot{\theta}_i}{m_i \times l_i} + g \times sin\left(\dot{\theta}_i\right)\right) \end{aligned} \tag{5}$$

$$x[t+1] = x[t] + \tau \times \dot{x}[t] \tag{6}$$

$$\dot{x}[t+1] = \dot{x}[t] + \tau \times \ddot{x}[t] \tag{7}$$

$$\theta[t+1] = \theta[t] + \tau \times \dot{\theta}[t] \quad (8)$$

$$\dot{\theta}[t+1] = \dot{\theta}[t] + \tau \times \ddot{\theta}[t] \quad (9)$$

In Equations (2)–(9), x represents the position of the cart, $\dot{x}$ denotes the velocity of the cart, θ_i is the angular position of the i-th pole, and $\dot{\theta}_i$ indicates the angular velocity of the i-th pole.

Analogously to [41,85], with the aim at fostering the discovery of robust solutions, each controller has been evaluated for eight episodes differing with respect to the initial state of the system. In particular, two experimental conditions were considered:

1. "Fixed Initial States" condition, in which controllers are evaluated by using the initial states provided in Table 3;
2. "Randomly Varying Initial States" condition: in this scenario, at each trial the initial states were randomized according to the ranges reported in Table 4.

Table 3. Values of the state variables (x, $\dot{x}$, θ_1, $\dot{\theta_1}$, θ_2, $\dot{\theta_2}$) used to initialize the different episodes (referred to as Ep_i, with i denoting the index of the episode) in the Fixed Initial States condition.

Variable	Ep_1	Ep_2	Ep_3	Ep_4	Ep_5	Ep_6	Ep_7	Ep_8
x	−1.944	1.944	0.0	0.0	0.0	0.0	0.0	0.0
$\dot{x}$	0.0	0.0	−1.215	1.215	0.0	0.0	0.0	0.0
θ_1	0.0	0.0	0.0	0.0	−0.10472	0.10472	0.0	0.0
$\dot{\theta_1}$	0.0	0.0	0.0	0.0	0.0	0.0	−0.135088	0.135088
θ_2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$\dot{\theta_2}$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 4. Ranges used to randomize the initial values of the state variables in the Randomly Varying Initial States condition.

Variable	Min	Max
x	−1.944	1.944
$\dot{x}$	−1.215	1.215
θ_1	−0.10472	0.10472
$\dot{\theta_1}$	−0.135088	0.135088
θ_2	−0.10472	0.10472
$\dot{\theta_2}$	−0.135088	0.135088

In both experimental settings, an evaluation episode ends when one of the following conditions is met:

- the episode lasts 1000 steps;
- the angle of the long pole goes out of the viable range (see Table 2);
- the angle of the short pole exceeds its range (see Table 2);
- the cart position is out of the track (see Table 2).

The performance (i.e., fitness) of the agent F is computed according to Equation (10):

$$F = \frac{1}{8} \sum_{i=1}^{8} f_i \quad (10)$$

where f_i—the fitness achieved by the agent in the i-th episode—is defined in Equation (11).

$$f_i = \frac{t}{1000} \quad (11)$$

In Equation (11), the variable t denotes the number of steps the cart successfully keeps both poles balanced. The evolutionary process is continued until the total number of performed steps exceeds 5×10^7.

As far as the Randomly Varying Initial States condition is concerned, differently from [41], this setup is used to verify the ability of the evolved controllers to deal with the Fixed Initial States condition. In other words, the random initial states are only used to make agents experience a large number of different conditions. This, in turn, should foster their robustness [85–87] and, therefore, their effectiveness at dealing with the initial states defined in Table 3. The reason behind such design choice depends on the consideration that the performance on the Randomly Varying Initial States condition is strongly biased by chance; some controllers might experience very easy situations to cope with and achieve high fitness scores, while others might suffer from hard initial conditions leading to final poor performances. Evaluating the evolving agents on the Fixed Initial States condition allows discriminating between truly effective and lucky controllers.

2.1.3. Optimization Functions

In this work, the Rastrigin [67], the Rosenbrock [68] and the Sphere optimization functions were considered, which represent widely employed benchmark functions in evolutionary computation and optimization [66,70,88,89]. Differently from the other considered problems, in this scenario, the goal is to minimize the value of the objective function. In particular, given a vector $x = [x_1, \ldots, x_n]$ of size n, the problem can be generically formulated as follows:

$$F = \min_{x \in \mathbb{R}^n} \left(F_{fname}(x) \right) \tag{12}$$

where $fname$ indicates the name of the function to be optimized (i.e., Rastrigin, Rosenbrock and Sphere).

The Rastrigin function is defined according to Equation (13):

$$F_{rastrigin} = \sum_{i=1}^{n} \left({x_i}^2 - 10 \times \cos(2\pi x_i) + 10 \right) \tag{13}$$

The Rosenbrock function can be formulated according to Equation (14):

$$F_{rosenbrock} = \sum_{i=1}^{n-1} \left(100 * \left(x_{i+1} - {x_i}^2 \right)^2 + (x_i - 1)^2 \right) \tag{14}$$

Finally, the Sphere function is defined according to Equation (15):

$$F_{sphere} = \sum_{i=1}^{n} {x_i}^2 \tag{15}$$

A visual representation of the optimization functions is provided in Figure 3. As can be seen, these functions are characterized by the existence of a single global optimum and multiple local minima (see Figure 3). Consequently, discovering an effective solution is far from trivial.

In the experiments reported in this work, individuals are evaluated and rewarded for the ability to optimize the above described functions on vectors x of size $n = 20$. Furthermore, the length of the evolutionary process was set to 10^6 evaluation steps.

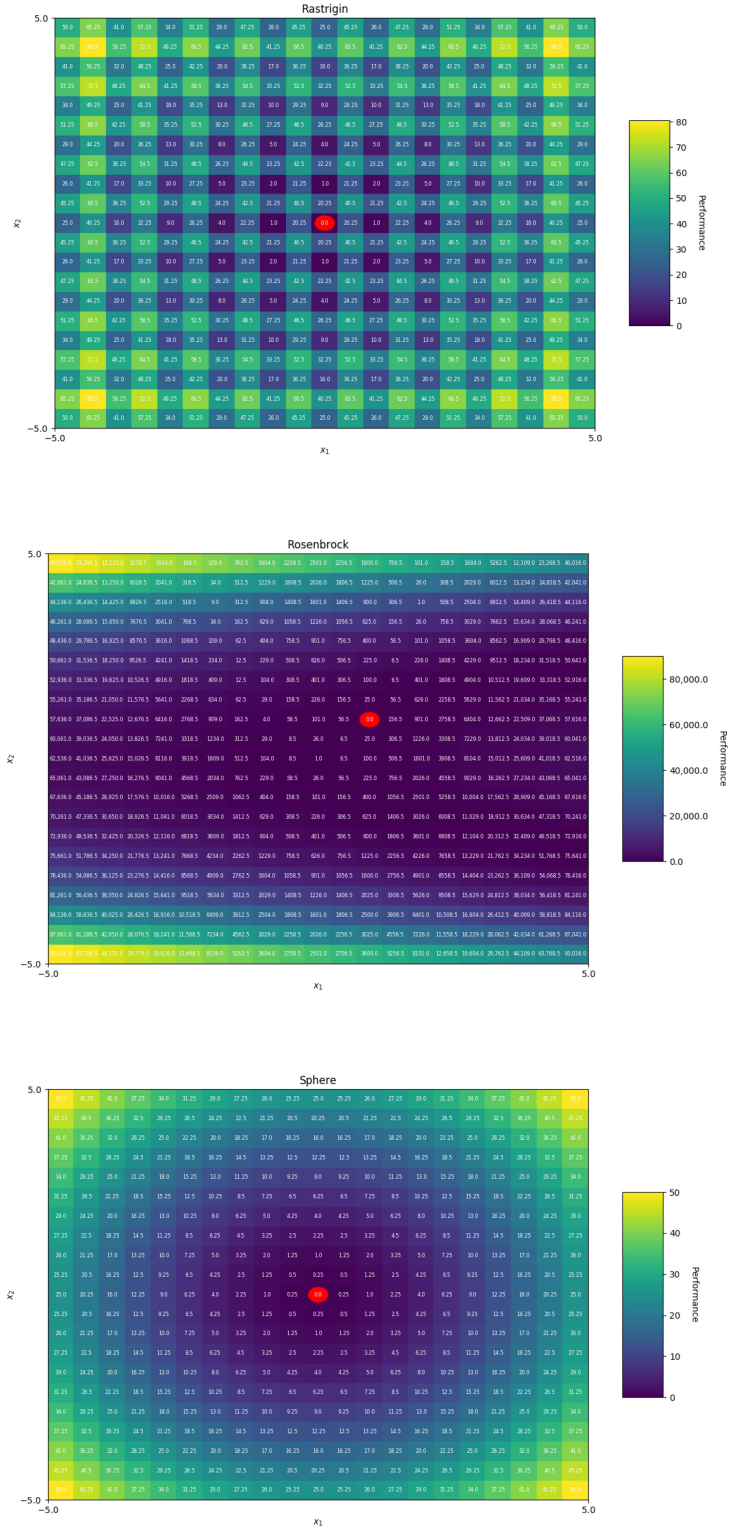

Figure 3. Two-dimensional visualization of the Rastrigin (**top**), Rosenbrock (**middle**) and Sphere (**bottom**) functions. To facilitate understanding, the case of a two-dimensional input $x = [x_1, x_2]$ is considered. The red circle within each colormap denotes the optimum of the corresponding function.

2.1.4. Robot Foraging

The robot foraging task is a modified version of the swarm-foraging problem [85,90], in which an e-puck robot [91] is situated in a squared arena of 2 m $\times$ 2 m surrounded by walls (see Figure 4). The environment contains five food elements. The goal for the agent is

to forage the highest number of food items. The fitness function F is defined as shown in Equation (16):

$$F = \frac{1}{5} \sum_{i=1}^{5} nf_i \tag{16}$$

where nf_i represents the number of food items eaten in the i-th episode. When the robot manages to eat one food element, this suddenly reappears in a different random location. The robot can perceive objects (i.e., food elements and walls) through a linear camera with a field of view (FOV) of 90° split into six sectors (see Figure 4). Each sector has three channels for red, green and, blue colors.

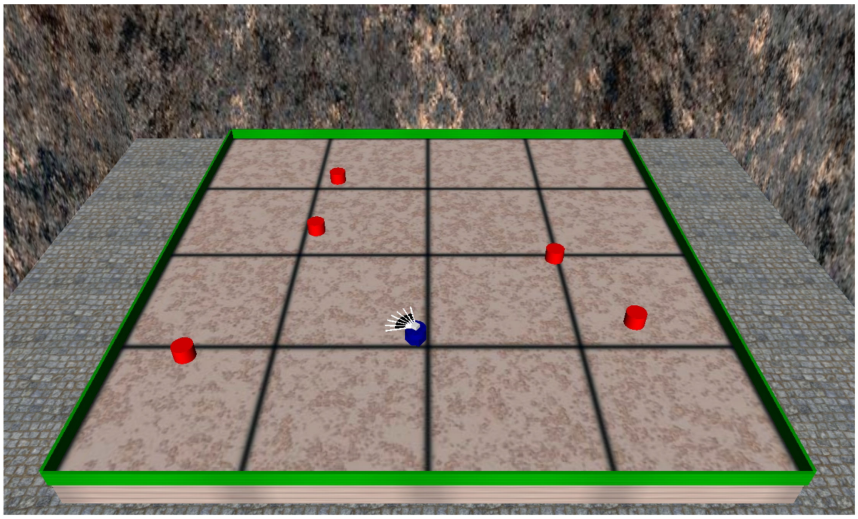

Figure 4. The robot foraging task. The robot (blue circle) lives in an environment surrounded by walls (in green) and filled with five food elements (red cylinders). Both the robot and food items are randomly placed in the environment.

The controller of the robot is a feed-forward neural network with 18 sensory inputs (provided by the camera), eight internal neurons and two output motors determining the speeds of the robot wheels. The network architecture was taken from similar experimental settings [71,72]. The evolutionary process lasts 5×10^6 evaluation steps. Individuals are evaluated in five episodes lasting 1000 steps.

2.1.5. Social Foraging

The social foraging problem is commonly used to analyze the complexity of dynamics and the emergent behaviors in systems where two agents might interact with each other [72,73]. In this work, the formulation reported in [72] was used, in which two e-puck robots are situated in a 2 m $\times$ 2 m arena surrounded by walls. The environment contains one food element (see Figure 5). The agents have to forage "socially", that is, they can eat the food item only if both manage to reach it. The goal for the robots is to forage the highest number of food items. The fitness function F rewards agents at a group level and is defined as for the robot foraging (see Equation (16)). It is worth pointing out that the definition of the fitness function F does not provide any information about how agents should behave. In particular, there is no constraint pushing the robots to arrive simultaneously at the food item. Effective solutions could entail that one agent moves toward the food element, reaching it and waiting for the mate. An alternative strategy for the robots might be to look for each other and reach the food item together.

Different from the work reported in [72,73], here, the two agents are homogeneous; the neural network controller is the same for both robots. This is due to the fact that the focus of this work is neither on the analysis of the group dynamics, nor on the ability to adapt to the partner skills. Instead, the aim is to understand the factors fostering a positive

influence of learning on evolution. The network structure is the same as defined for the robot foraging task. Similarly, the length of the evolutionary process and the evaluation of each individual are the same as used in the robot foraging problem.

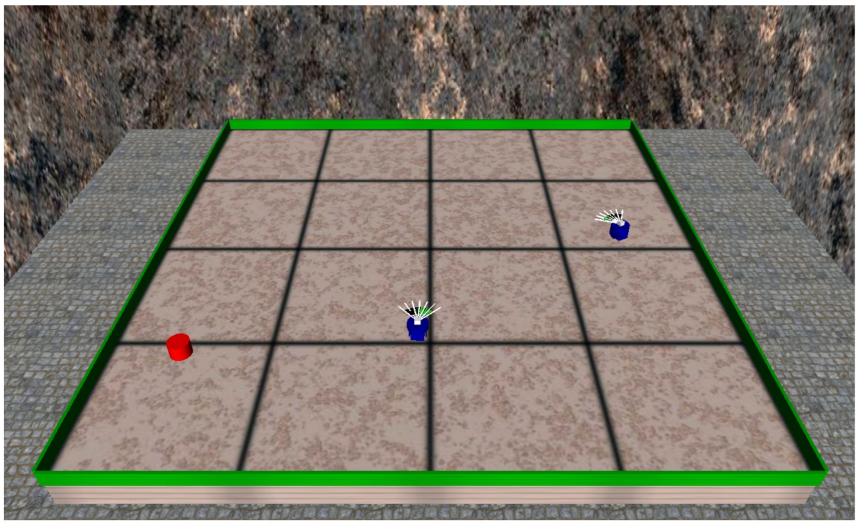

Figure 5. The social foraging task. The robots (blue circles) live in an environment surrounded by walls (in green) and filled with one food element (red cylinder). Both the robots and food item are randomly placed in the environment.

2.2. Evolutionary Algorithms

Before focusing on the evolutionary algorithms used in this work, it is important to underline the different ways of initializing the population depending on the nature of the evolutionary problem. Concerning the double-pole balancing task, the optimization of the Rastrigin, Rosenbrock and Sphere functions, the robot foraging problem and the social foraging task, the initial population is encoded in a matrix of $\mu \times \theta$ integer numbers randomly initialized with a uniform distribution in the range $[0, 255]$ and then converted into values in the range $[-8.0, 8.0]$ with respect to the double-pole balancing problem, and in the range $[-5.0, 5.0]$ for both the function optimization scenario and the robot foraging and the social foraging tasks. In particular, μ corresponds to the number of parents and θ corresponds to the number of neural network's parameters (i.e., connection weights and biases). Offspring are obtained from their corresponding parents by mutating each gene with a $MutRate$ probability. In particular, when a mutation is performed, the original gene is replaced with a new integer number randomly generated within the range $[0, 255]$ with a uniform distribution. On the other hand, in the case of the 5-bit parity task, the population contains individuals whose genotype is constituted by a vector of integer numbers that encode the logic function calculated by each gate and the way gates are wired. This method is called Cartesian Genetic Programming (CGP) [77]. More specifically, each genotype includes $400 \times 3 = 1200$ genes specifying the characteristics of the nodes and one additional gene representing the identification number of the node that is the output of the circuit. The indices of the inputs and the indices of the nodes belong to the ranges [1–5] and [6–406], respectively. Each node is made of three genes:

1. inp_1: index of the first input of the node. It is bounded in the range $[1, 5 + (L - 1) \times 20]$, where L denotes the layer of the gate;
2. inp_2: index of the second input of the node. It belongs to the same range as inp_1;
3. $func$: integer representing the logic function of the node (1 = OR, 2 = AND, 3 = NAND, 4 = NOR).

The value of the last gene encoding the index of the output belongs to the range $[6, 406]$. Mutations (with probability $MutRate$) are made by substituting each gene with a value randomly extracted from a uniform distribution in the appropriate range.

2.2.1. Stochastic Steady State

The first evolutionary algorithm considered in this work is a variant of the standard steady-state evolutionary algorithm [92], called Stochastic Steady State [41].

The Stochastic Steady State (SSS) is a $(\mu + \mu)$ evolutionary strategy [93,94] operating on the basis of a population formed by μ parents. The pseudo-code of the algorithm is provided in Algorithm 1. At each generation, parents are evaluated (Algorithm 1, line 7) and generate one offspring each (Algorithm 1, lines 9–10), which, in turn, are evaluated (Algorithm 1, line 12). The individuals are then ranked based on their fitness and the best μ ones are retained in the population (see Algorithm 1, line 14). Differently from previous related methods, like the Steady State [92], in [41], the authors introduced the possibility of adding noise to the fitness, thus making the selective process stochastic. The noise is a value randomly chosen in the range $[-NoiseRange, NoiseRange]$ with a uniform distribution and is applied to the individual's fitness according to Equation (17):

$$\begin{aligned} Noise &\sim rand(-NoiseRange, NoiseRange) \\ \hat{F} &= F \times (1.0 + Noise) \end{aligned} \tag{17}$$

Therefore, the application of noise enables shaping the selection process [41]. In fact, the higher the noise, the higher the probability that less fit individuals reproduce. Differently from the original algorithm, the version used in this work (see Algorithm 1) was modified by removing the solution refinement during the last $\frac{1}{20}$ period of the evolutionary process (see [41] (pp. 10–11)).

Algorithm 1 SSS algorithm

Functions: init(), eval(), mut()

1: Initialize: $NEvals \leftarrow 0$, $PopSize$
2: **for** $p \leftarrow 0$ to $PopSize$ **do**
3: $init(genome[p])$ ▷ Population is randomly initialized
4: **end for**
5: **while** $NEvals < MaxEvals$ **do**
6: **for** $p \leftarrow 0$ to $PopSize$ **do**
7: $Fitness[p] \leftarrow eval(genome[p])$
8: $NEvaluations \leftarrow NEvaluations + NSteps$
9: $genome[p + NParents] \leftarrow genome[p]$ ▷ Create a copy of the parents' genotype, i.e., offspring
10: $mut(genome[p + NParents])$ ▷ Mutate the genotype of the offspring
11: $Fitness[p + NParents] \leftarrow eval(genome[p + NParents])$
12: $NEvals \leftarrow NEvals + NSteps$
13: **end for**
14: **Rank** *genotype* for *Fitness*
15: **end while**

2.2.2. Hill-Climbing

The Hill-Climbing (HC) algorithm is a method introduced in [49], which evolves solutions through a local search process. A description of the method is given in Algorithm 2. As for the SSS algorithm, the population is formed by μ parents. At each generation, each parent generates an offspring, i.e., a mutated copy (Algorithm 2, lines 9–10), and both individuals are evaluated (Algorithm 2, lines 7 and 11). If the offspring is not worse than its parent (Algorithm 2, line 13), the former replaces the latter in the population (Algorithm 2, line 14). Therefore, differently from the SSS, each individual of the population evolves independently.

Algorithm 2 HC algorithm

Functions: init(), eval(), mut()

1: Initialize: $NEvals \leftarrow 0$, $PopSize$
2: **for** $p \leftarrow 0$ to $PopSize$ **do**
3: $init(genome[p])$ ▷ Individuals are randomly initialized
4: **end for**
5: **while** $NEvals < MaxEvals$ **do**
6: **for** $p \leftarrow 0$ to $PopSize$ **do**
7: $Fitness[p] \leftarrow eval(genome[p])$
8: $NEvals \leftarrow NEvals + NSteps$
9: $genome[p + NParents] \leftarrow genome[p]$ ▷ Create a copy of the parents' genotype, i.e., offspring
10: $mut(genome[p + NParents])$ ▷ Mutate the genotype of the offspring
11: $Fitness[p + NParents] \leftarrow eval(genome[p + NParents])$
12: $NEvals \leftarrow NEvals + NSteps$
13: **if** $Fitness[p + NParents] > Fitness[p]$ **then**
14: $genome[p] \leftarrow genome[p + NParents]$
15: $Fitness[p] \leftarrow Fitness[p + NParents]$
16: **end if**
17: **end for**
18: **end while**

2.2.3. Stochastic Steady State with Hill-Climbing

The Stochastic Steady State with Hill-Climbing (SSSHC) algorithm is a novel method developed by the author (similar works can be found in [48,60]), which combines the SSS algorithm with a particular Lamarckian learning process implemented through a stochastic Hill Climber [49]. The former method performs the evolutionary process by selecting the fittest individuals in the population. On the other hand, the latter technique refines the solution found by applying a local search process, which retains adaptive mutations producing an improvement. These modifications are inherited in the population, thus HC acts as Lamarckian learning. As stated in Section 1, the SSSHC belongs to the family of memetic algorithms. Algorithm 3 provides the pseudo-code of the SSSHC algorithm; it works as the SSS method until the selection process (Algorithm 3, lines 1–14), and learning is applied to selected individuals only. During the latter phase (Algorithm 3, lines 15–28), the individuals undergo a refinement process for a fixed number of iterations $NumLearnIters$ (Algorithm 3, lines 18–27). At each learning iteration, the currently selected individual is mutated (as it happens during offspring generation, see Algorithm 3, lines 10 and 20) and the novel individual, referred as the *candidate*, is evaluated (Algorithm 3, line 21). The currently selected individual is then compared with the candidate (Algorithm 3, line 23). The best performing individual is retained as the currently selected individual. The process is repeated until the given number of learning iterations has been run.

2.2.4. Parameters

As already pointed out in Section 1, the algorithms used in this work were compared by taking into account the same number of evolutionary steps and with the same parameter settings. More specifically, in the 5-bit parity task and the double-pole balancing problem, the best combination of parameters $MutRate$ and $PopSize$ for the SSS method is determined, which is referred to as "BestSetup" and represents the baseline. The HC and SSSHC algorithms were evaluated only by using parameters of the "BestSetup". The rationale behind this choice lies in the need for a fair comparison so as to exclude any possible bias due to the use of different parameters. It is worth pointing out that this approach might clearly penalize the HC and SSSHC strategies, since there is no guarantee that the optimal parameters found for a method are the same for all the considered algorithms. On the other hand, with regard to both the function optimization scenario and the robot foraging and

social foraging problems, the parameters used for the SSS, HC and SSSHC algorithms were not varied and are reported in Tables 5 and 6, respectively.

Algorithm 3 SSSHC algorithm

Functions: init(), eval(), mut()

Initialize: $NEvals \leftarrow 0$, $PopSize$, $NLearnIters$
for $p \leftarrow 0$ to $PopSize$ **do**
  $init(genome[p])$ ▷ Population is randomly initialized
end for
while $NEvals < MaxEvals$ **do**
  for $p \leftarrow 0$ to $PopSize$ **do**
    $Fitness[p] \leftarrow eval(genome[p])$
    $NEvaluations \leftarrow NEvaluations + NSteps$
    $genome[p + NParents] \leftarrow genome[p]$ ▷ Create a copy of the parents' genotype, i.e., offspring
    $mut(genome[p + NParents])$ ▷ Mutate the genotype of the offspring
    $Fitness[p + NParents] \leftarrow eval(genome[p + NParents])$
    $NEvals \leftarrow NEvals + NSteps$
  end for
  Rank *genotype* for *Fitness*
  for $p \leftarrow 0$ to $PopSize$ **do**
    $selectedPop[p] \leftarrow genome[p]$
    $selectedFit[p] \leftarrow Fitness[p]$
    for $iter \leftarrow 0$ to $NLearnIters$ **do**
      $candidate \leftarrow selectedPop[p]$
      $mut(candidate)$
      $candidateFit \leftarrow eval(candidate)$
      $NEvals \leftarrow NEvals + NSteps$
      if $candidateFit > selectedPop[p]$ **then**
        $selectedPop[p] \leftarrow candidate$
        $selectedFit[p] \leftarrow candidateFit$
      end if
    end for
  end for
end while

Table 5. List of parameters used in the function optimization scenario. As already stated in the text, the size of the input vector x was set to $n = 20$.

Parameter	Value
$NumReplications$	20
$MutRate$	0.05
$PopSize$	20

Table 6. List of parameters used in the robot foraging and the social foraging problems. They were derived from similar experimental settings (see [72,73]).

Parameter	Value
$NumReplications$	20
$MutRate$	0.01
$PopSize$	20

Four of the five considered problems were analyzed in two experimental conditions: in the "No noise" condition, the agent evaluation is not affected by the addition of noise.

Instead, a certain amount of noise is applied to the performance measure in the "Noisy" condition, thus influencing the evolutionary process. With respect to the 5-bit parity task and the double-pole balancing problem, the best value of the *NoiseRange* parameter for the SSS method was used to evaluate also the HC and SSSHC algorithms, similarly to the *MutRate* and *PopSize* parameters. Conversely, the parameter *NoiseRange* was set to 0.05 in the "Noisy" condition of the robot foraging and the social foraging problems. Finally, the optimization of the Rastrigin, Rosenbrock and Sphere functions was performed only in the "No noise" condition.

3. Results

In this section, the results obtained in the five different scenarios are presented and discussed (an overview of the outcomes is shown in Figure 6). The metrics used to evaluate the different algorithms are the performance/fitness, which measures the ability of the discovered solutions to cope with the given problem, and the convergence speed, i.e., the number of evaluation episodes required to find the optimal solution to the task. The latter was calculated only for the 5-bit parity task and the Fixed Initial States condition of the double-pole balancing problem. Indeed, the optimal value of the Rastrigin, Rosenbrock and Sphere functions was not discovered by the considered algorithms (see Section 3.3). Moreover, the maximum performance that can be achieved in the robot foraging and social foraging problems is not known in advance. With respect to the statistical analyses, the Mann–Whitney U test with application of the Bonferroni correction was used. A value of p below 0.05 indicates statistical significance. Conversely, when the compared conditions are equivalent, the statistical analysis returns a value of p above 0.05. Moreover, the Spearman's Rank correlation coefficient (denoted with ρ) was employed to analyze the relationships between variables, and the associated statistical significance (p below 0.01) is provided where applicable.

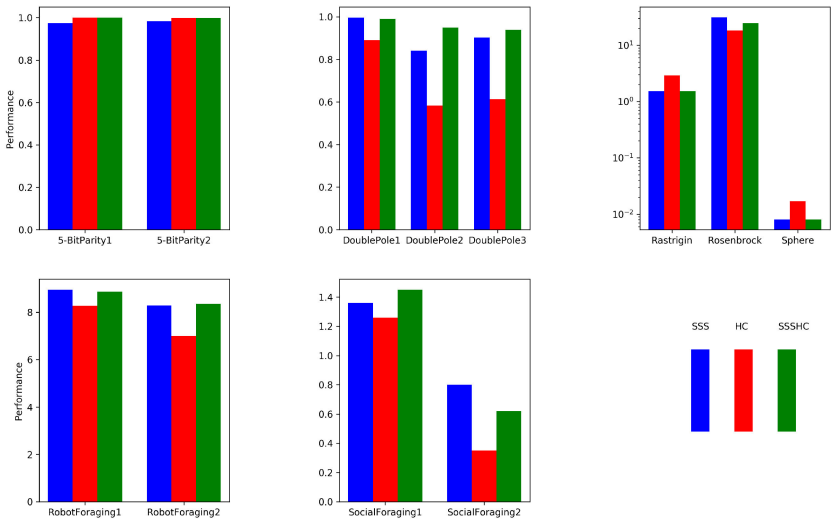

Figure 6. Performance of the SSS, the HC and the SSSHC algorithms with respect to the different considered evolutionary problems. Task labels in the x-axis are defined as follows: 5-BitParity1: 5-bit parity, "No noise" condition; 5-BitParity2: 5-bit parity, "Noisy" condition; DoublePole1: double-pole balancing, Fixed Initial States condition, "No noise" case; DoublePole2: double-pole balancing, Fixed Initial States condition, "Noisy" case; DoublePole3: double-pole balancing, Randomly Varying Initial States condition; RobotForaging1: robot foraging, "No noise" condition; RobotForaging2: robot foraging, "Noisy" condition; SocialForaging1: social foraging, "No noise" condition; SocialForaging2: social foraging, "Noisy" condition. With respect to the optimization function scenario (top right figure), performance in the y-axis is displayed through a logarithmic scale and a lower fitness corresponds to a better result.

3.1. 5-Bit Parity

As indicated by the results reported in Table 7, the SSSHC and HC algorithms outperform the SSS method in the deterministic case ($p < 0.05$). Furthermore, they require a considerably lower number of evaluations ($p < 0.05$, see Table 8). Noticeably, the SSSHC algorithm is significantly faster than the HC method ($p < 0.05$, see Table 8). The same result holds with the addition of noise with respect to both performance ($p < 0.05$, see Table 7) and convergence speed ($p < 0.05$, see Table 8). In the "Noisy" condition, the HC algorithm is slightly better than the SSSHC algorithm, although the difference is not statistically significant ($p > 0.05$). Interestingly, in the deterministic case, the SSSHC algorithm manages to solve this relatively complex task with $NumLearnIters \in [100, 10,000]$ (see Table A3), although it achieves a remarkable performance of 0.99 even with 20–50 learning iterations (see Table A3). The addition of noise makes the task harder to be solved, requiring a number of learning iterations higher than 200 to achieve a performance of 0.99 (see Table A3).

Table 7. Average fitness of the controllers evolved with the SSS, the HC and the SSSHC algorithms in the 5-bit parity task. Data were obtained by running 50 replications of the experiment. Data in square brackets indicate the standard deviation. All the algorithms were evaluated by using the best combination of parameters for the SSS in the "No noise" condition. The following parameter values were used: $MutRate = 1\%$; $PopSize = 10$ (see Table A1). The *NoiseRange* parameter (i.e., "Noisy" condition) was set to 0.03 (see Table A2). Regarding the SSSHC, the number of learning iterations *NumLearnIters* was set to 2000 in the "No noise" condition and 5000 in the "Noisy" condition, respectively (see Table A3). Bold values denote the best results.

	SSS	**HC**	**SSSHC**
No noise	0.974 [0.053]	**1.0 [0.0]**	**1.0 [0.0]**
Noisy	0.983 [0.037]	**0.999 [0.009]**	0.998 [0.018]

Table 8. Average number of evaluations required to find a solution to the 5-bit parity problem by the controllers evolved with the SSS, the HC and the SSSHC algorithms. Data represent the result of 50 replications of the experiment. The parameter settings are the same as reported in the caption of Table 7. Bold values indicate the best results.

	SSS	**HC**	**SSSHC**
No noise	5.766×10^7	8.802×10^6	$\mathbf{5.433 \times 10^6}$
Noisy	5.003×10^7	2.599×10^7	$\mathbf{1.731 \times 10^7}$

Figure 7 displays the performance obtained by the different algorithms during evolution. As can be seen, SSSHC has a convergence speed noticeably faster than SSS and faster than HC. Concerning the "No noise" case (Figure 7, top), SSSHC manages to find a quasi-optimal solution (i.e., a performance $F \geq 0.95$) after 5.225×10^6 evaluation steps, HC succeeds in 7.324×10^6 evaluation steps, while SSS requires 5.295×10^7 evaluation steps, i.e., more than 10 times the convergence speed of SSSHC and more than seven times the convergence speed of HC. With respect to the "Noisy" case (Figure 7, bottom), SSSHC discovers a quasi-optimal solution after 1.73×10^7 evaluation steps, HC achieves a fitness score over 0.95 in 2.091×10^7 evaluation steps, while SSS requires 4.565×10^7 evaluation steps, corresponding to more than 2.5 times the convergence speed of SSSHC and more than two times the convergence speed of HC. The difference is statistically significant in both conditions ($p < 0.05$).

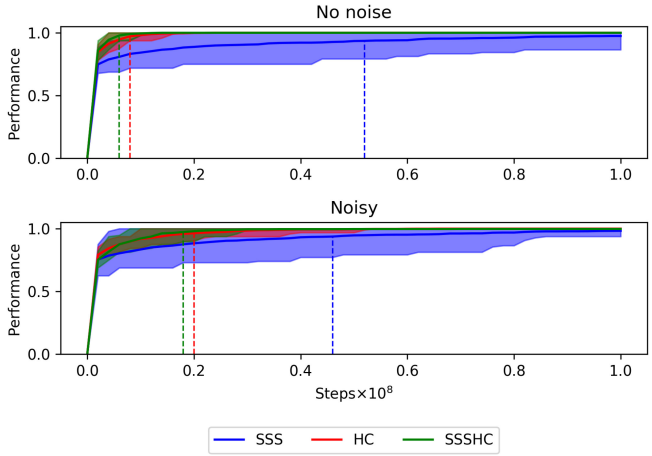

Figure 7. Performance of the SSS, the HC and the SSSHC algorithms with the best combination of parameters (*MutRate* = 1% , *PopSize* = 10) in the 5-bit parity task. Top picture refers to experiments without the addition of noise (*NumLearnIters* = 2000). Bottom picture shows results obtained with the addition of noise (*NoiseRange* = 0.03; *NumLearnIters* = 5000). The shadow areas indicate the mean and 85% bootstrapped confidence intervals of the mean. The vertical dashed line marks the number of steps required by the algorithms to achieve a quasi-optimal solution (i.e., a performance score greater or equal to 0.95). These data were achieved by averaging outcomes from 50 replications of the experiment.

Figure 8 shows the performance of the SSSHC algorithm depending on the number of learning iterations. The data indicate that the SSSHC algorithm is effective at finding a solution independently of the length of the learning process, at least in this context. Nonetheless, there exists a positive correlation between fitness and number of learning iterations (ρ = 0.330699, significant at $p < 0.01$ in the "No noise" condition; $\rho = 0.243837$, significant at $p < 0.01$ in the "Noisy" condition). Furthermore, the analysis of the correlation between the number of evaluation steps and the number of learning iterations displays a positive correlation (ρ = 0.549272, significant at $p < 0.01$ in the "No noise" condition; $\rho = 0.380872$, significant at $p < 0.01$ in the "Noisy" condition). This implies that, at least in this domain, the longer the learning process, the faster the algorithm.

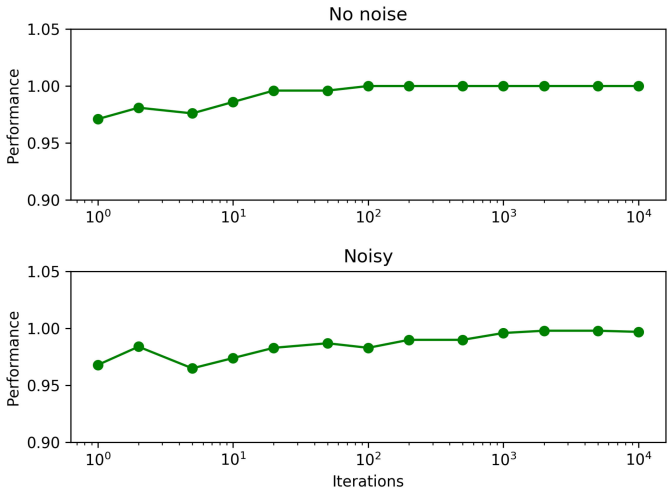

Figure 8. Analysis of the performance of the SSSHC algorithm in the 5-bit parity task depending on the different number of learning iterations. Circles indicate the fitness values obtained with the corresponding number of iterations. The top curve shows results without the application of noise, while the bottom one displays the performance with the addition of noise (*NoiseRange* = 0.03). Data in the x-axis are shown using a logarithmic scale and indicate the average performance over 50 replications of the experiment.

3.2. Double-Pole Balancing

With respect to the double-balancing task, the Markovian version of the problem was not considered, since it is quite easy to solve. The results obtained in this domain indicate that the SSS, HC and SSSHC algorithms successfully manage to find the optimal solution to the problem in both the deterministic ("No noise") and the stochastic case ("Noisy").

3.2.1. Fixed Initial States Condition

As indicated by the results reported in Table 9, the SSS and SSSHC algorithms outperform the HC algorithm ($p < 0.05$) in both conditions. Moreover, the former methods are noticeably faster than the latter in the "No noise" condition ($p < 0.05$, see Table 10). The performance and convergence speed of the SSSHC and SSS algorithms are similar ($p > 0.05$). Considering the addition of noise ("Noisy" condition), the SSSHC remarkably outperforms the SSS algorithm ($p < 0.05$, see Table 9), while there is no significant difference concerning the convergence speed ($p > 0.05$). The outcomes are further illustrated in Figure 9.

Table 9. Average fitness of the controllers evolved with the SSS, the HC and the SSSHC algorithms in the Fixed Initial States condition of the double-pole balancing problem. These data were obtained by repeating the experiment 30 times. Data in square brackets indicate the standard deviation. All the algorithms were evaluated by using the best combination of parameters for the SSS in the "No noise" condition. The mutation rate *MutRate* was set to 5% and the population size *PopSize* is 200 (see Table A4); the *NoiseRange* parameter (i.e., "Noisy" condition) was set to 0.06 (see Table A5). The number of learning iterations of the SSSHC algorithm was set to 1 in the "No noise" condition and to 50 in the "Noisy" condition (see Table A6). Bold values indicate the best results.

	SSS	**HC**	**SSSHC**
No noise	**0.996 [0.015]**	0.891 [0.081]	0.991 [0.028]
Noisy	0.840 [0.100]	0.584 [0.143]	**0.950 [0.052]**

Table 10. Average number of evaluations required to find a solution to the Fixed Initial States condition of the double-pole balancing problem by the controllers evolved with the SSS, the HC and the SSSHC algorithms. Data correspond to the outcome of 30 replications of the experiment. The parameter settings are the same as reported in the caption of Table 9. Bold values indicate the best results.

	SSS	**HC**	**SSSHC**
No noise	2.81×10^7	4.795×10^7	$\mathbf{2.458 \times 10^7}$
Noisy	4.702×10^7	5×10^7	$\mathbf{4.386 \times 10^7}$

The results obtained in the non-Markovian version of the double-pole suggest a possible relationship between the learning performance and the population size. Differently from the 5-bit parity and the Markovian version of the task (results not shown), in this case, the optimal population size is 200, while in the former tasks, the optimal value is 10. The hypothesis is that using a large population size does not allow exploring the search space effectively and finding of optimal solutions. Aiming to verify this assumption, a control experiment was run, where only the population size was varied and all other parameters were kept fixed (see Table 11). As a reference task, the non-Markovian double-pole with Fixed States initial condition was used. The mutation rate *MutRate* was set to 0.05. The number of learning iterations *NumLearnIters* was set to 5. No noise (i.e., *NoiseRange* = 0.0) was added to the learning process. The population size was varied by using the following values: $PopSize \in [10, 20, 50, 100, 200, 500]$. Overall, 30 replications of the experiment were performed, thus evaluating 26400 individuals. The analysis revealed a negative correlation between the fitness of the individuals and the population size ($\rho = -0.788579$, significant at $p < 0.01$). In other words, the larger the population size, the lower the performance

of the individuals being part of it. This result confirms the hypothesis about the negative effect of large population sizes on the learning process, at least in this domain.

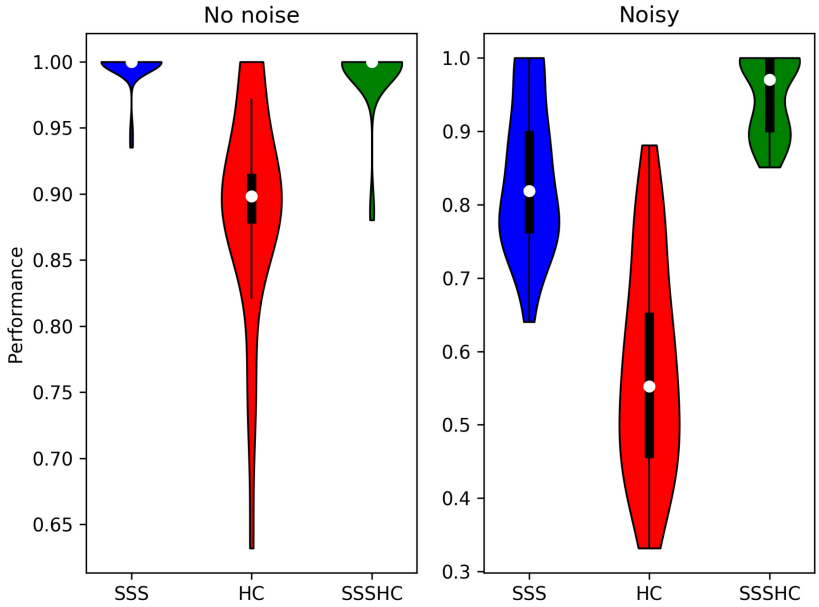

Figure 9. Fitness of the SSS, the HC and the SSSHC algorithms in the Fixed Initial States condition of the double-pole balancing problem. The black rectangles inside violins contain the inter-quartile range of the data, while the median value of the data is indicated by the white circle inside each violin. The extreme points of the vertical black line denote the minimum and maximum values of the data. Results achieved in 30 replications of the experiment by using the best combination of parameters (mutation rate: 5%; population size: 200). Left figure refers to the "No noise" condition ($NumLearnIters$ = 1), while the right plot corresponds to the "Noisy" condition ($NoiseRange$ = 0.06; $NumLearnIters$ = 50).

Table 11. List of parameters used to verify the existence of a correlation between learning and population size. The latter parameter was set as $PopSize \in [10, 20, 50, 100, 200, 500]$.

Parameter	Value
NumReplications	30
MutRate	0.05
NoiseRange	0.0
NumLearnIters	5

Analogously to the 5-bit parity problem, the correlation between performance and number of learning iterations was investigated. The outcome of this analysis indicates that there exists a negative correlation in the "No noise" condition (ρ = - 0.285502, significant at $p < 0.01$), while fitness and duration of learning positively correlate in the "Noisy" condition (ρ = 0.290081, significant at $p < 0.01$). This discrepancy (see also Figure 10) can be ascribed to both the nature of the evolutionary problem and the role played by noise; differently from the 5-bit parity task, the double-pole balancing problem is not characterized by neutrality (see Section 4). Instead, it is highly sensitive to small changes. Therefore, exploring the search space for a long period may be disadvantageous, since learning might not be able to improve the solution at the cost of reducing the number of remaining evaluation steps. This property is stronger in a deterministic scenario, like the "No noise" condition. Conversely, in the "Noisy" condition, learning exploits the addition of noise in order to discover adaptive mutations and escape from local optima solutions. In

this case, a longer learning process is crucial to substantially improve the fitness. The latter outcome is in line with the analysis of the 5-bit parity task, where a positive correlation between performance and duration of the learning process was found.

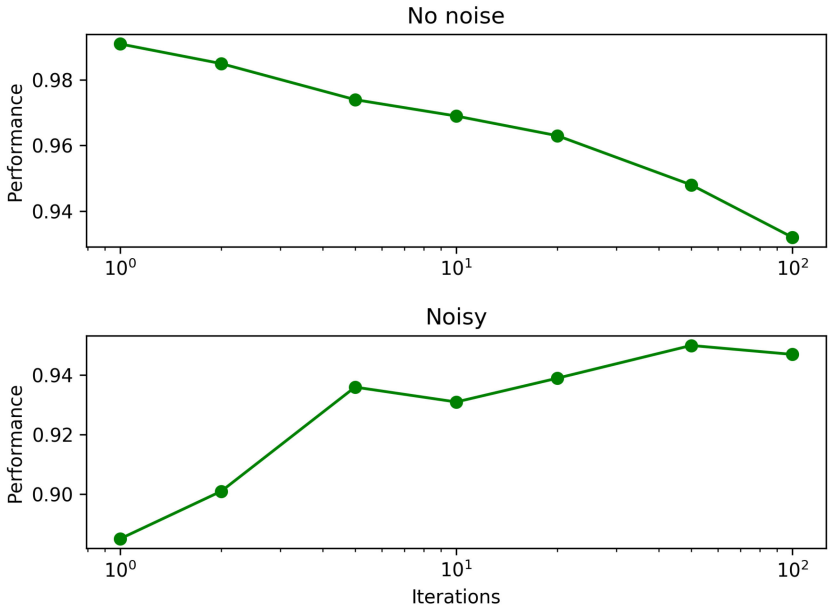

Figure 10. Fitness obtained by the SSSHC algorithm in the Fixed Initial States condition of the double-pole balancing problem by systematically varying the number of learning iterations. Circles indicate the performances obtained with the corresponding number of iterations. As can be observed, performance decreases as the number of learning iterations increases in the "No noise" condition (top figure). Conversely, there exists a positive correlation between the number of learning iterations and fitness in the "Noisy" condition (bottom figure), with $NoiseRange = 0.06$. Data in the x-axis are shown using a logarithmic scale and were achieved by running 30 replications of the experiment.

Concerning the relationship between the number of evaluation steps and the length of the learning process, there is a negative correlation both in the "No noise" condition ($\rho = -0.450261$, significant at $p < 0.01$) and in the "Noisy" condition ($\rho = -0.082322$), although not significant in the latter case. Put in other words, reducing the length of the learning process leads to better and faster results in this domain.

3.2.2. Randomly Varying Initial States Condition

Due to the intrinsic stochasticity of the considered scenario, only the "No noise" case was investigated with regard to the Randomly Varying Initial States condition. In particular, as has been previously explained, the ability of the evolved controllers to perform well in the Fixed Initial States condition was analyzed. In this respect, the results reported in Table 12 and Figure 11 indicate that the SSS and SSSHC algorithms remarkably outperform the HC algorithm ($p < 0.05$). Furthermore, the SSSHC is better than the SSS, although the difference is not statistically significant ($p > 0.05$). Therefore, learning provides a positive effect on the search process, driving it towards higher fitness areas of the search space. It is worth noting that the capability to "generalize" (i.e., to display good performance in the Fixed Initial States condition) is achieved in spite of the intrinsic variability of the task. Further analyses should demonstrate whether or not this property can be extended to other domains.

Table 12. Average fitness of the controllers evolved with the SSS, the HC and the SSSHC algorithms in the Randomly Varying Initial States condition of the double-pole balancing problem. Data indicate the performance obtained by evaluating evolved solutions in the Fixed Initial States condition and are the result of 30 replications of the experiment. Data in square brackets indicate the standard deviation. Data refer to the best combination of parameters ($MutRate = 5\%$; $PopSize = 50$) (see Table A7). The SSSHC performance refers to the best case ($NumLearnIters = 2$) (see Table A8). Bold values indicate the best results.

SSS	HC	SSSHC
0.903 [0.076]	0.614 [0.159]	**0.939 [0.066]**

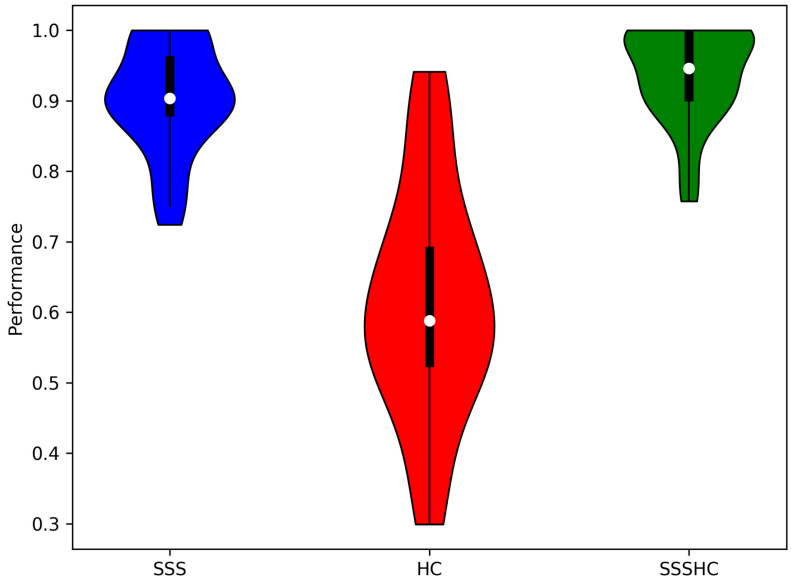

Figure 11. Performance of the SSS, the HC and the SSSHC algorithms in the Randomly Varying Initial States condition of the double-pole balancing problem. Data represent the fitness achieved by evaluating evolved controllers in the Fixed Initial States condition. Violins show the distribution of data. The black rectangles inside violins are bounded in the range $[Q_1, Q_3]$, where Q_1 denotes the first quartile and Q_3 indicates the third quartile. The white circle inside each violin represents the median value of the data. The vertical black line is limited between the minimum and maximum values of the data. These outcomes were achieved through 30 replications of the experiment by using the best combination of parameters (mutation rate: 5%; population size: 50).

Finally, in this case, the correlation between the performance and number of learning iterations was also analyzed. The results show the existence of a negative correlation ($\rho = -0.16011$, significant at $p < 0.01$), i.e., the longer the learning process, the worse the performance. This outcome is due to the low neutrality of the considered case, which is characterized by high variance (the task is stochastic). Consequently, the learning process mainly generates maladaptive perturbations while reducing the number of remaining evaluation steps.

3.3. Optimization Functions

The analysis of the performance in the function optimization scenario demonstrates the superiority of the SSS and SSSHC algorithms over the HC method in two of the considered problems. Specifically, SSS and SSSHC significantly outperform HC in optimizing the Rastrigin and Sphere functions ($p < 0.05$). Interestingly, the former methods achieve the same performance (see Table 13). Instead, with respect to the Rosenbrock function, SSSHC is remarkably superior to HC ($p < 0.05$) and is also better than SSS (see Table 13), although the latter outcome is not statistically significant ($p > 0.05$).

Table 13. Average fitness of the controllers evolved with the SSS, the HC and the SSSHC algorithms with respect to the Rastrigin, Rosenbrock and Sphere optimization functions. Data were collected by replicating the experiments 20 times. Data in square brackets indicate the standard deviation. The values of the mutation rate *MutRate* and population size *PopSize* used in both conditions are reported in Table 6. The SSSHC performance refers to the best case found for all functions ($NumLearnIters = 1$ concerning the Rastrigin and Sphere functions, and $NumLearnIters = 100$ with respect to the Rosenbrock function, see Tables A9–A11). Bold values indicate the best results (lower is better).

	SSS	HC	SSSHC
Rastrigin	**1.524 [0.0]**	2.891 [0.668]	**1.524 [0.0]**
Rosenbrock	31.496 [27.841]	18.473 [3.612]	**16.119 [2.100]**
Sphere	**0.008 [0.0]**	0.017 [0.004]	**0.008 [0.0]**

The obtained results indicate that, in this context, the addition of learning to evolution does not help to improve the quality of the discovered solutions in the optimization of the Rastrigin and Sphere functions (see Table 13). Conversely, learning is slightly beneficial with regard to the optimization of the Rosenbrock function. This discrepancy can be partly ascribed to the difference in the global optimum of the considered functions: both the Rastrigin and the Sphere functions are minimized when $x = [0, \ldots, 0]$, with x denoting the input vector (see Equations (13) and (15)). Instead, the Rosenbrock function is optimized when the $x = [1, \ldots, 1]$ (see Equation (14)). Overall, finding such types of solutions when inputs are not binary and are converted into floating-point values is not trivial, since it is easy to become stuck in local optima (see Figure 3). Moreover, the order of magnitude of Rosenbrock is notably higher than those of the Rastrigin and Sphere functions (see Table 13).

Figure 12 displays the performance variation during the evolutionary process. As can be seen, the SSS converges to lower values before the HC and SSSHC with respect to both the Rastrigin and Sphere functions (see Figure 12, top and bottom figures). On the other hand, the three algorithms exhibit similar trends when optimizing the Rosenbrock function (Figure 12, middle figure).

By looking at the correlation between the value of the function and the number of learning iterations, a positive correlation was found with regard to both the Rastrigin ($\rho = 0.887625$, significant at $p < 0.01$) and Sphere ($\rho = 0.937437$, significant at $p < 0.01$) functions. This implies that increasing the length of the learning process is detrimental, since performance strongly decreases (recall that, in this domain, the goal is to minimize the target function). Conversely, there exists a negative correlation in the case of the Rosenbrock function ($\rho = -0.818182$, significant at $p < 0.01$). Differently from the other two functions, here, making the learning process longer is beneficial for the discovery of more effective solutions (see Figure 13).

As already stated in Section 2.2.4, the experimental parameters were kept fixed (see Table 5) and not systematically varied. Therefore, it is possible that using different settings might lead to improved performances for all the considered algorithms, especially with regard to the Rosenbrock function. The results (not shown) indicate that the performance of the SSSHC method can increase when diverse values of the *MutRate* and *PopSize* parameters are used. Moreover, the addition of noise was not investigated in this scenario. Consequently, introducing noise in the selection process could help escape from the local minima solutions discovered by the different algorithms.

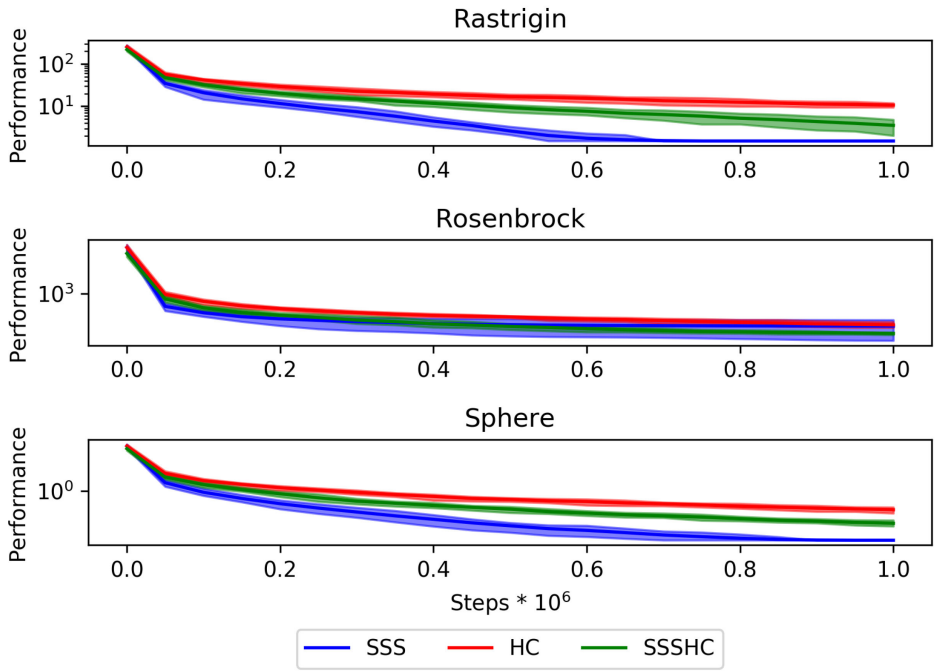

Figure 12. Performance of the SSS, the HC and the SSSHC algorithms in the optimization of the Rastrigin (**top figure**), Rosenbrock (**middle figure**) and Sphere (**bottom figure**) functions. The shadow areas indicate the mean and 85% bootstrapped confidence intervals of the mean. The performance on the y-axis is displayed by means of the logarithmic scale. Collected data are the average result of 20 replications of the experiments. Parameters that were used are reported in Table 5.

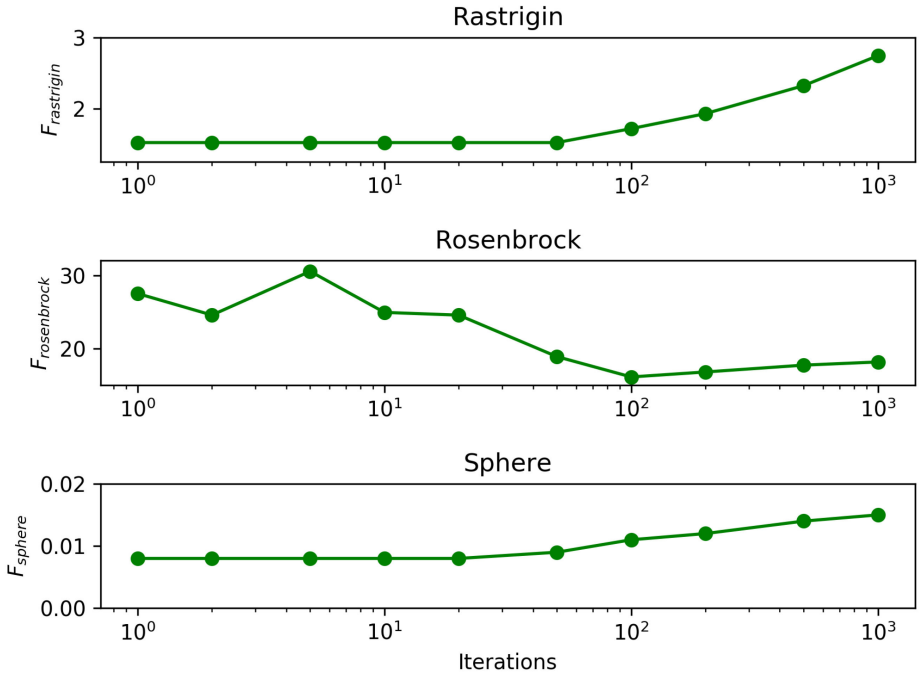

Figure 13. Values of the Rastrigin ($F_{rastrigin}$), Rosenbrock ($F_{rosenbrock}$) and Sphere (F_{sphere}) functions achieved by the SSSHC algorithm by systematically varying the number of learning iterations. Circles denote the fitness obtained with the corresponding number of iterations. With respect to the Rastrigin and Sphere functions, the value of the function is higher as the number of learning iterations increases (**top and bottom figures**). Conversely, as far as the Rosenbrock function is concerned, the value of the function decreases when the number of learning increases (**middle figure**). Data in the x-axis are shown using a logarithmic scale and were collected in 20 replications of the experiment.

3.4. Robot Foraging

The performance comparison of the considered methods in the robot foraging task generates results in line with those found in the function optimization scenario. In fact, as indicated in Table 14 and Figure 14, the SSS and SSSHC algorithms remarkably outperform the HC method ($p < 0.05$). Moreover, the SSS algorithm achieves a slightly higher performance than the SSSHC method in the "No noise" condition, while the opposite is true in the "Noisy" condition. However, the difference is not statistically significant in both cases ($p > 0.05$). Therefore, the combination of learning and evolution does not provide an advantage to evolution in this scenario. Moreover, the addition of noise does not help to improve the quality of solutions (see Table 14). Instead, it has a detrimental effect on performance, which is significantly lower ($p < 0.05$). Some insights explaining why the addition of learning to evolution is not beneficial are provided in Section 4.

Table 14. Performance of the controllers evolved with the SSS, the HC and the SSSHC algorithms in the robot foraging task. These data were obtained by averaging the results of 20 replications of the experiment. Standard deviations are reported within square brackets. The parameters *MutRate* and *PopSize* were set according to Table 6. Regarding the "Noisy" condition, the parameter *NoiseRange* was set to 0.05, as stated in Section 2.2.4. The SSSHC performance refers to the best case found for both conditions (*NumLearnIters* = 1 in the "No noise" condition and *NumLearnIters* = 2 in the "Noisy" condition, see Table A12). Bold values indicate the best results.

	SSS	**HC**	**SSSHC**
No noise	**8.950 [0.315]**	8.270 [0.386]	8.870 [0.305]
Noisy	8.290 [0.392]	6.990 [0.816]	**8.350 [0.460]**

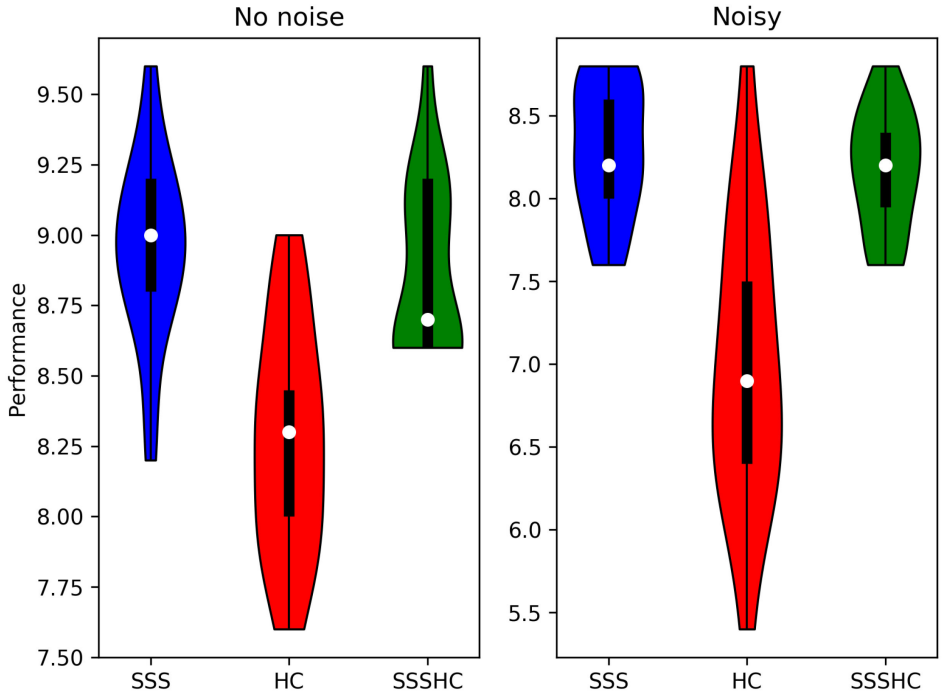

Figure 14. Performance of the SSS, the HC and the SSSHC algorithms in the robot foraging task. The black rectangles contained in the violins are bounded in the range $[Q_1, Q_3]$. Median values are represented by white circles. The vertical black line is limited between the minimum and maximum values of the data. Data were obtained by averaging 20 replications of the experiments. Parameters that were used are reported in Table 6. Left figure refers to the "No noise" condition (*NumLearnIters* = 1), while right picture refers to the "Noisy" condition (*NoiseRange* = 0.05; *NumLearnIters* = 2).

It is important to note that, differently from the 5-bit parity and the double-pole balancing problems, here, the parameter settings were not systematically varied, analogously to the function optimization scenario. Instead, they were set according to the values reported in Table 6, which are derived from works focusing on the analysis of dynamics and emergent behaviors in groups [71,72]. Similarly, the value of the parameter *NoiseRange* was set to 0.05 in the "Noisy" condition, without any additional analysis. Therefore, there could be room for further performance improvements of the SSS, HC and SSSHC algorithms. In the future, the investigation of different combinations of parameters will be explored.

Figure 15 shows how the performance of the considered methods varies during the evolutionary process. Since the curves are increasing (see Figure 15), continuing the process is highly likely to further improve the fitness of the different algorithms.

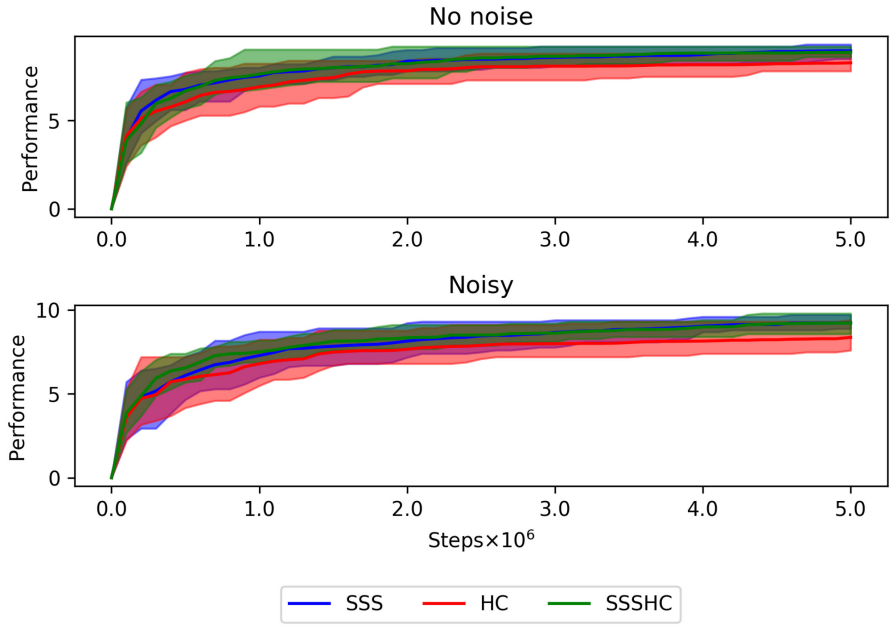

Figure 15. Performance of the SSS, the HC and the SSSHC algorithms in the robot foraging task. Top picture refers to the "No noise" condition, while the bottom figure contains data achieved in the "Noisy" condition (*NoiseRange* = 0.05). With respect to the SSSHC algorithm, data refer to the best value of the parameter *NumLearnIters* found (*NumLearnIters* = 1 in the "No noise" condition, *NumLearnIters* = 2 in the "Noisy" condition). The shadow areas indicate the mean and 85% bootstrapped confidence intervals of the mean. These data were obtained by averaging the outcomes from 20 replications of the experiment.

3.5. Social Foraging

The SSSHC algorithm achieves better performance then the HC method in both conditions of the social foraging problem (see Table 15 and Figure 16). The result is statistically significant ($p < 0.05$). Moreover, the SSS is considerably superior to HC in the "Noisy" condition ($p < 0.05$, see Table 15). The SSS and SSSHC perform similarly in this domain, with no significant differences among them ($p > 0.05$). These outcomes are coherent with those found for the robot foraging task and confirm that learning is not beneficial for evolution when the considered problem entails agent–environment interactions. Indeed, the complexity of these domains makes it difficult for learning to locally refine solutions, since variations mainly have a disruptive effect. Consequently, the SSSHC is likely to undergo phases in which learning performs additional evaluation episodes aiming to improve the performance of current solutions, but without success.

Table 15. Fitness of the controllers evolved with the SSS, the HC and the SSSHC algorithms in the social foraging problem. Data represent the average performance of 20 replications of the experiment. Square brackets contain the standard deviations. As for the robot foraging task, the parameters *MutRate* and *PopSize* that were used are reported in Table 6. Moreover, these results were achieved using the best value of the parameter *NumLearnIters* found for both conditions (*NumLearnIters* = 1 in the "No noise" condition and *NumLearnIters* = 5 in the "Noisy" condition, see Table A13). Bold values indicate the best results.

	SSS	**HC**	**SSSHC**
No noise	1.360 [0.250]	1.260 [0.329]	**1.450 [0.218]**
Noisy	**0.800 [0.410]**	0.350 [0.252]	0.620 [0.166]

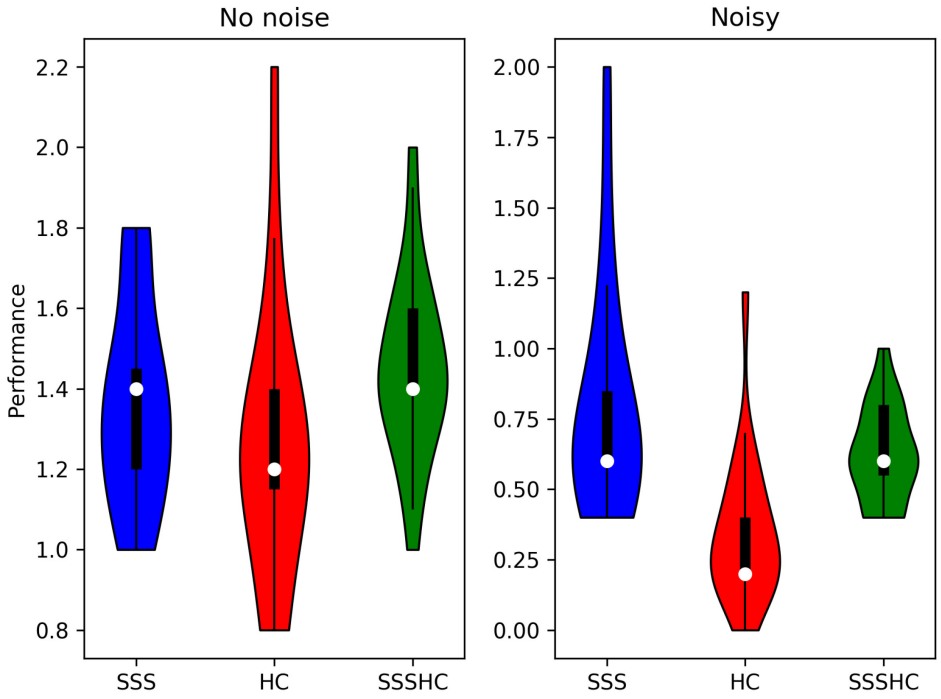

Figure 16. Performance of the SSS, the HC and the SSSHC algorithms in the social foraging problem. The black rectangles inside violins contain the inter-quartile range of the data. White circles inside each violin denote the median. The extreme points of the vertical black line indicate the minimum and maximum values of the data. These data are the result of 20 replications of the experiments. Parameters that were used are reported in Table 6. Left figure refers to the "No noise" condition (*NumLearnIters* = 1), while right picture refers to the "Noisy" condition (*NoiseRange* = 0.05; *NumLearnIters* = 5).

Notably, in this scenario, the addition of noise is highly detrimental, with a remarkable drop in performance (see Table 15). In fact, the fitness achieved in the "Noisy" condition is considerably lower than the one obtained in the "No noise" condition ($p < 0.05$, see also Figure 16). This outcome is in line with the results found in the robot foraging task, although the drop in performance is less marked in the latter case (see Table 14). However, the social foraging problem is considerably more challenging than the robot foraging task, because the performance strongly depends on the ability of the agents to forage "socially", i.e., the solution to the problem cannot be achieved by a single robot. This entails that both agents should have exploratory skills, find the food element and eat the item. Despite the use of homogeneous agents, which makes the task easier, the need to explore the environment in searching for the food element could prevent the robots from behaving effectively. This also explains why the performance in the robot foraging task is about one order of magnitude higher than the fitness achieved in the social foraging problem

(see Tables 14 and 15). Given the complexity of the considered problem, the addition of noise has a disruptive effect on the performance of solutions discovered by the considered algorithms. Indeed, only a few mutations have a positive effect on fitness, but noise hinders such adaptation mechanisms and affects both the selection and the learning processes. In this respect, future studies should clarify whether a limited amount of noise could be beneficial in the considered scenario.

Analogously to the robot foraging task, the analysis of fitness during the whole evolutionary process indicates that the different algorithms can further improve their results (see Figure 17). This is emphasized more for the SSS algorithm, especially in the "Noisy" condition (see Figure 17, bottom). Consequently, it is possible that increasing the length of the evolutionary process could demonstrate that learning is detrimental in this scenario.

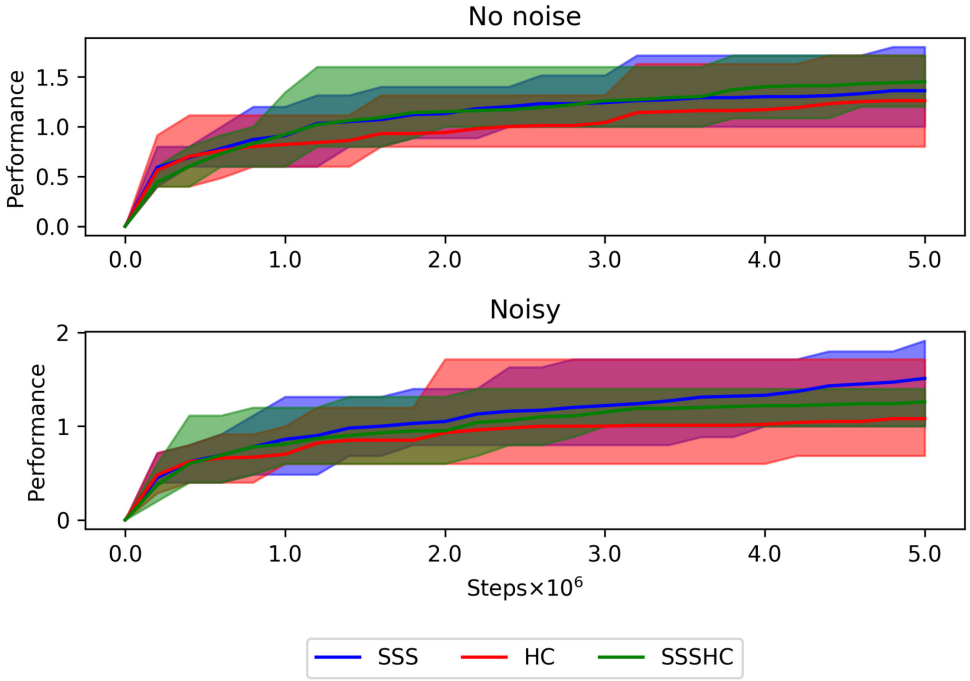

Figure 17. Average fitness obtained by the SSS, the HC and the SSSHC algorithms in the social foraging problem. Top picture refers to the "No noise" condition, while bottom figure contains data achieved in the "Noisy" condition (*NoiseRange* = 0.05). The outcomes of the SSSHC algorithm refer to the best value of the parameter *NumLearnIters* found (*NumLearnIters* = 1 in the "No noise" condition, *NumLearnIters* = 5 in the "Noisy" condition). The shadow areas indicate the mean and 85% bootstrapped confidence intervals of the mean. These data are the results of 20 replications of the experiment.

4. Discussion

In this section, the outcomes of the experiments reported in Section 3 and the main findings are discussed.

With respect to the 5-bit parity task, the better performance of HC over SSS is not surprising. As pointed out in [76], $(1 + \lambda)$-ES algorithms like HC (where $\lambda = 1$) display higher performance than $(\mu + \mu)$-ES techniques, such as SSS, in this domain. Indeed, the family of parity problems is characterized by high neutrality, i.e., large areas of the search space that can be reached through mutations not affecting the probability of an individual surviving and reproducing [95,96]. Therefore, the former methods drive evolution towards such neutral regions of the search space that can ultimately lead to areas with higher fitness. Conversely, the latter algorithm tends to explore regions of the search space characterized by high robustness but far from high-fitness areas [76]. The competition between population members observed in $(\mu + \mu)$-ES algorithms results in the tendency

to perform a local exploration of the search space, thus preventing them from discovering optimal solutions to the problem. Moreover, solutions discovered by $(1 + \lambda)$-ES methods typically contain a higher number of genes playing a functional role than those found by $(\mu + \mu)$-ES techniques [76]. The SSSHC algorithm exploits the combination of the two different techniques in order to achieve very good performance (see Table 7) with considerably better convergence speed (see Table 8).

Concerning the double-pole balancing problem, the HC is significantly worse than SSS. This implies that strategies in which competition among population members does not play a role are not effective at finding a solution to the task. The possibility to compete against other individuals allows avoiding becoming stuck in sub-optimal solutions. The outcome is even more evident in the "Noisy" condition. Indeed, the addition of noise has a disruptive effect on the HC algorithm, since the possibility of retaining maladaptive traits increases. Because most of the mutations typically cause a drop in performance and given that evolving individuals do not compete for survival, the combination of these two factors leads to the impossibility to access areas of the search space corresponding to higher fitness. Conversely, the SSS method is less sensitive to the issue, due to the competition among population members triggered by the selection process. The SSSHC benefits from the latter property to avoid being trapped in local minima, while preserving the capability to explore and, possibly, improve the quality of the discovered solutions. Overall, the obtained results confirm the hypothesis about the positive influence of noise on learning. Indeed, making the fitness stochastic allows learning to explore more of the search space. The retention of maladaptive mutations gives learning the possibility to access areas of the search space that cannot be reached through a deterministic process.

The performance of SSS and SSSHC is superior to that of HC with respect to the optimization of the Rastrigin and Sphere functions. Furthermore, the SSSHC considerably outperforms the HC and is better than the SSS when optimizing the Rosenbrock function. Overall, these outcomes show how the combination of learning and evolution is not advantageous in this scenario. One possible explanation is related to the goal of these types of problem, i.e., finding the best set of values that minimize the considered function. As illustrated in Section 3.3, this implies the discovery of global optima solutions (i.e., $x = [0, \ldots, 0]$ for the Rastrigin and Sphere functions, $x = [1, \ldots, 1]$ for the Rosenbrock function, respectively), which is far from trivial. In addition, the presence of multiple local optima solutions makes it difficult for the algorithms to improve performance. In this respect, mutations are mainly maladaptive and improve performance only rarely. Consequently, learning spends time in trying to discover more effective solutions and fails in providing a beneficial effect on evolution. As illustrated in Section 3.3, the experiments on function optimization were performed without considering the addition of noise, which could help to escape from the local optima solutions characterizing the considered functions, similarly to what has been observed in the double-pole balancing problem. Therefore, further analysis is needed before reaching a final conclusion.

The SSS and SSSHC algorithms discover more effective solutions than the HC method in both the robot foraging and the social foraging tasks. Moreover, the performance of the SSS and the SSSHC is similar, without a clear winner. Consequently, learning does not provide an advantage to evolution in this domain. The main reason lies in the diverse nature of the considered problems. In fact, differently from the 5-bit parity and double-pole balacing tasks, these problems involves robotic agents exploring the arena in search of food elements to eat. Consequently, the agents interact with the environment. The SSS exploits the competition between individuals in the population in order to avoid being stuck in local minima solutions and obtains a relatively good fitness on average. On the other hand, the HC algorithm maintains different individuals evolving in parallel, but it fails in achieving adequate performance. The SSSHC benefits from the combination of the two approaches, but the time spent to refine solutions does not provide an advantage to evolution in this context. As pointed out with respect to the function optimization scenario, it is worth considering that there is no guarantee that learning improves the quality of the solutions

discovered during the evolutionary process, since most of the mutations are maladaptive. Consequently, the SSSHC might be characterized by phases in which learning does not play a role. Because the local search is costly in terms of number of evaluation episodes, SSSHC achieves performances comparable to those obtained by SSS in this domain. The properties of the SSSHC algorithm also clarify why the addition of noise is detrimental in this scenario (see Table 15), as already explained in Section 2.1.5.

One might argue that the double-pole balancing problem and the two considered robotic tasks share similarities, since both have agents interacting with the environment (e.g., the cart and the robots). Nevertheless, the types of interaction are completely different: in the double-pole balancing problem, the cart can only move on the horizontal axis and the interaction with the environment implies the contact of the wheels with the floor (see Figure 2), without friction. On the other hand, in both the robot foraging and the social foraging tasks, agents are embodied and can freely move in the environment. Moreover, they perceive objects in the nearby areas (e.g., food elements and/or walls), which affect their behaviors. This implies that dealing with these stimuli is remarkably more complex. Finally, the actions performed by the robots cause sudden changes in the inputs; for example, when the agent succeeds in eating a food item, this reappears immediately in a different location, and the sensory inputs of the robot are significantly different between one step and the next (see Table 16).

Table 16. Example of sensory inputs perceived by the robot in two consecutive steps of the robot foraging problem. Specifically, step t contains the values of the sensors just before eating a food element, while step $t+1$ refers to the inputs after the food item has been eaten. Sensors are grouped by color channel (red: R1–R6, green: G1–G6, blue: B1–B6), as described in Section 2.1.4. As can be seen, the stimuli received by the agent undergo a sudden change due to the fact that the eaten food element reappears in a different location.

Step	R1	R2	R3	R4	R5	R6	G1	G2	G3	G4	G5	G6	B1	B2	B3	B4	B5	B6
t	0.0	0.0	0.933257	1.0	1.0	0.798138	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$t+1$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

As shown in Figures 15 and 17, all the considered algorithms might improve their performances if the evolutionary process continues. This is more evident when evaluating the SSS in the "Noisy" condition of the social foraging experiment. Therefore, future work will investigate the performances of the different methods when the length of the evolutionary process is set to 5×10^8 evaluation steps as in [72] and whether the effect of learning on evolution becomes detrimental.

5. Conclusions

In this paper, the benefits and drawbacks of combining evolution and learning, a well-known topic in the research community, were investigated. Prior works in this area have led to contradictory results, without providing clear hints about the actual effect of learning on evolution. Differently from previous approaches, results were collected in five different scenarios, including three benchmark tasks (5-bit parity, double-pole balacing and function optimization) and two robotic problems (robot foraging and social foraging). The aim is to investigate this interplay on widely used and challenging domains. The hypothesis is that learning provides advantages to evolution under specific conditions, like the use of deterministic episodes and the absence of agent–environment interactions. When these criteria are met, the addition of noise to both selection and learning process is beneficial. Here, the term "learning" denotes a refinement process attempting to modify parameters through mutations and retaining adaptive modifications. A novel algorithm combining learning and evolution, called SSSHC, was proposed and tested on the above mentioned tasks. Specifically, the author compared the SSSHC algorithm with the methods composing it, namely, the SSS algorithm (i.e., a pure evolutionary method) and the HC

algorithm, which represents an optimization algorithm performing solution refinements based. The three techniques were analyzed by keeping the same parameter settings to avoid biases: the best parameters for the SSS algorithm are first determined and used to evaluate both the HC and the SSSHC algorithms. The results in the presented domains indicate that the combination of evolution and learning may or may not be beneficial depending on the nature of the problem. In particular, the following outcomes can be highlighted:

- the SSSHC achieves significantly better performance than the SSS with respect to the 5-bit parity task (both "No noise" and "Noisy" conditions);
- the SSSHC algorithm is notably faster than the other methods with regard to the 5-bit parity task (both "No noise" and "Noisy" conditions);
- the SSSHC considerably outperforms the SSS in to the Fixed Initial States condition, "Noisy" case of the double-pole balancing problem;
- the SSSHC is better than SSS in the Randomly Varying Initial States condition of the double-pole balancing problem;
- the SSSHC is remarkably superior to HC with respect to both conditions ("No noise" and "Noisy") of the double-pole balancing problem;
- the SSSHC has a significantly superior convergence speed compared to the HC in the Fixed Initial States condition of the double-pole balancing problem, "No noise" case;
- the performance of SSS and SSSHC is the same with respect to the optimization of the Rastrigin and Sphere functions;
- the SSSHC significantly outperforms the HC in the optimization of the Rastrigin and Sphere functions;
- the SSSHC is notably superior to HC in the optimization of the Rosenbrock function;
- the SSSHC is better than SSS in the optimization of the Rosenbrock function;
- the SSSHC performs similarly to SSS in both the robot foraging task and the social foraging problem;
- the SSSHC is remarkably superior to HC in both the robot foraging task and the social foraging problem;
- the SSSHC has a considerable performance drop when noise is added in both the robot foraging task and the social foraging problem.

Moreover, the achieved results reveal that the advantage is higher when noise is added to the learning and the selection processes if the considered problem is characterized by determinism, i.e., when evolving agents are evaluated in the same episodes throughout the whole evolutionary process. Indeed, the possibility to retain maladaptive traits in order to explore more of the search space allows the discovery of areas of higher fitness that cannot be reached through a standard evolutionary process. However, some differences can be highlighted: in the 5-bit parity task, the length of the learning process positively correlates with both the final performance and the convergence speed. The result was obtained in both the "No noise" condition and the "Noisy" condition. This implies that the effect of learning on evolution is higher when the number of learning iterations increases. An explanation lies in the nature of the parity problem, which is characterized by high neutrality. This, in turn, allows the learning process to efficiently explore the search space and ultimately discover the regions associated to high performance. With regard to the double-pole balancing problem, instead, the duration of the learning process is negatively correlated with both the fitness and the convergence speed in the "No noise" case of the Fixed Initial States condition. Differently from the 5-bit parity task, the double-pole balancing problem is characterized by low neutrality. Consequently, small changes might have disruptive effects. This explains why reducing the length of the learning process produces better results. Conversely, in the "Noisy" case of the Fixed Initial States condition, the effect of learning on evolution is more beneficial when the number of learning iterations is higher. Indeed, the addition of noise could allow learning to escape from local optima and generate more effective solutions. Overall, the collected results clearly indicate the need to fine-tune the duration of the learning process based on the specific evolutionary

problem. In the future, the possibility of adjusting the number of learning iterations during the course of the evolutionary process should be investigated.

On the other hand, learning does not positively affect evolution on the function optimization scenario, the robot foraging and social foraging problems. Specifically, with regard to the former case, the Rastrigin, Rosenbrock and Sphere functions are characterized by one global optimum and multiple local minima, which make the task challenging. However, the analysis reveals differences between the examined functions: with respect to the Rastrigin and Sphere functions, the performance is negatively correlated with the length of the learning process, as for the "No noise" case of the Fixed Initial States condition of the double-pole balancing task. Conversely, the fitness is positively correlated with the duration of learning in the optimization of the Rosenbrock function, analogously to the 5-bit parity task and the "Noisy" case of the Fixed Initial States condition of the double-pole balancing problem. In addition, it is worth highlighting that the function optimization scenario was investigated only in the absence of noise. As already discussed in Section 4, future studies should clarify whether and how the addition of noise might influence the interplay between learning and evolution in this domain.

As far as the robot foraging and social foraging problems are concerned, the embodiment and the presence of agent–environment interactions increase the task complexity. The addition of noise in the robot foraging and social foraging problems has a detrimental effect on the performance of the considered algorithms, due to the nature and the complexity of these tasks. Nonetheless, further exploration of the effect of noise and other parameter settings is required, since these problems were studied by using the values reported in Tables 5 and 6, without performing any systematic analysis. Similarly, a future research direction could explore the possibility of automatically adjusting the parameters (i.e., mutation rate and population size) on the fly, depending on the performance level during the evolutionary process.

It is worth underlining that the learning definition used in this work does not correspond to the classic concept reported in [26]. Consequently, in the future, the author plans to study the relationship between evolution and learning more thoroughly, by investigating learning techniques like back-propagation [97] or Spike Timing-Dependent Plasticity (STDP) [98,99]. Furthermore, future works will be devoted to verify whether the outcomes reported here remain valid by using automatic algorithm configuration tools to optimize parameters, a topic beyond the scope of this paper as pointed out in Section 1. In addition, the results reported in this work show that, when agents actively interact with the environment, learning does not provide an advantage to evolution. Future studies will investigate possible modifications of the SSSHC (for example, the utilization of a crossover operator or the adaptation of the length of learning on the fly) in order to achieve better performance than the SSS in the considered experimental settings, as well as in other scenarios involving autonomous robots performing individual [100–102] or collective behaviors [103–106]. Finally, future research will be devoted to the comparison of the SSSHC with other state-of-the-art EAs like the Covariance Matrix Adaptation Evolution Strategy (CMA-ES, see [64,107]), Separable Natural Evolution Strategies (sNES, see [66,108]) Exponential Natural Evolution Strategies (xNES, see [66,109]) and the OpenAI Evolutionary Strategy (OpenAI-ES, see [110–112]), and/or with Reinforcement Learning (RL) methods like the Proximal Policy Optimization (PPO) [113] and Deep Deterministic Policy Gradient (DDPG) [114].

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The code to replicate the experiments reported in the paper is freely available at https://github.com/PaoloP84/LearningAndEvolution (accessed on 28 October 2024).

Acknowledgments: The author would like to express his gratitude to his friend and colleague Alessandra Vitanza for her suggestions that contributed to improving the quality of this work. The author would also like to thank Stefano Nolfi for his advice about the structure and organization of the article.

Conflicts of Interest: The author declares no conflicts of interest.

Appendix A

Appendix A.1. 5-Bit Parity

Table A1. Average fitness of the controllers evolved with the SSS algorithm in 50 replications of the experiment. Data in square brackets indicate the standard deviation. Data in round brackets represent the average number of evaluation episodes performed. Bold values indicate the best results.

MutRate/Pop Size	10	20	50	100
0.01	**0.974 [0.053]** (5.766×10^7)	0.939 [0.077] (7.838×10^7)	0.915 [0.075] (8.424×10^7)	0.880 [0.106] (8.191×10^7)
0.02	0.973 [0.059] (5.46×10^7)	0.933 [0.077] (7.463×10^7)	0.922 [0.071] (7.973×10^7)	0.906 [0.073] (8.584×10^7)
0.05	0.955 [0.052] (7.572×10^7)	0.917 [0.077] (8.334×10^7)	0.896 [0.081] (8.831×10^7)	0.898 [0.086] (8.67×10^7)
0.1	0.856 [0.083] (9.448×10^7)	0.832 [0.076] (9.808×10^7)	0.843 [0.090] (9.432×10^7)	0.824 [0.074] (9.896×10^7)
0.2	0.747 [0.059] (10^8)	0.729 [0.047] (10^8)	0.723 [0.039] (10^8)	0.716 [0.038] (10^8)

Table A2. Average fitness of the controllers evolved with the SSS algorithm with the best combination of parameters ($MutRate = 0.01$; $PopSize = 10$, see Table A1). Different levels of noise were applied. Data were obtained by running 50 replications of the experiment. Data in square brackets indicate the standard deviation. Data in round brackets represent the average number of evaluation episodes performed. Bold values indicate the best results.

NoiseRange	Fitness
0.0	0.974 [0.053] (5.766×10^7)
0.01	0.969 [0.063] (5.277×10^7)
0.02	0.969 [0.062] (5.497×10^7)
0.03	**0.983 [0.037]** (5.003×10^7)
0.04	0.976 [0.054] (5.335×10^7)
0.05	0.960 [0.070] (5.663×10^7)
0.06	0.969 [0.061] (5.612×10^7)
0.07	0.971 [0.052] (5.334×10^7)
0.08	0.951 [0.072] (5.98×10^7)
0.09	0.965 [0.063] (5.785×10^7)
0.1	0.964 [0.070] (5.338×10^7)
0.15	0.958 [0.067] (7.167×10^7)
0.2	0.945 [0.069] (6.647×10^7)

Table A3. Average fitness of the controllers evolved with the SSSHC algorithm. Experiments were run with the best combination of parameters found for the SSS algorithm ($MutRate = 0.01$; $PopSize = 10$, see Table A1). Experiments were run both without noise and with the best amount of noise found for the SSS algorithm ($NoiseRange = 0.03$, see Table A2). These data were achieved by replicating the experiment in 50 replications. Data in square brackets indicate the standard deviation. Data in round brackets represent the average number of evaluation episodes performed. Bold values indicate the best results.

Learning Iterations	No Noise	Noisy
1	0.971 [0.059] (5.455×10^7)	0.968 [0.054] (5.344×10^7)
2	0.981 [0.036] (5.065×10^7)	0.984 [0.034] (5.307×10^7)
5	0.976 [0.049] (4.898×10^7)	0.965 [0.064] (5.138×10^7)
10	0.986 [0.044] (3.942×10^7)	0.974 [0.051] (5.289×10^7)
20	0.996 [0.020] (2.998×10^7)	0.983 [0.035] (4.738×10^7)
50	0.996 [0.015] (1.889×10^7)	0.987 [0.042] (3.963×10^7)
100	1.0 [0.0] (1.365×10^7)	0.983 [0.039] (3.728×10^7)
200	1.0 [0.0] (9.908×10^6)	0.990 [0.034] (3.132×10^7)
500	1.0 [0.0] (5.736×10^6)	0.990 [0.025] (2.703×10^7)
1000	1.0 [0.0] (5.8×10^6)	0.996 [0.015] (2.105×10^7)
2000	**1.0 [0.0]** (5.433×10^6)	0.998 [0.014] (2.197×10^7)
5000	1.0 [0.0] (5.546×10^6)	**0.998 [0.018]** (1.731×10^7)
10000	1.0 [0.0] (6.331×10^6)	0.997 [0.013] (1.906×10^7)

Appendix A.2. Double-Pole Balancing

Appendix A.2.1. Fixed Initial States Condition

Table A4. Average fitness of the controllers evolved with the SSS algorithm. Data are the result of 30 replications of the experiment. Data in square brackets indicate the standard deviation. Data in round brackets represent the average number of evaluation episodes performed. Bold values indicate the best results.

MutRate/PopSize	10	20	50	100	200	500
0.01	0.088 [0.069] (5×10^7)	0.187 [0.228] (5×10^7)	0.420 [0.356] (4.324×10^7)	0.515 [0.336] (4.838×10^7)	0.699 [0.295] (4.494×10^7)	0.799 [0.159] (4.888×10^7)
0.02	0.261 [0.287] (5×10^7)	0.400 [0.345] (4.604×10^7)	0.664 [0.329] (4.625×10^7)	0.856 [0.250] (3.497×10^7)	0.926 [0.121] (4.271×10^7)	0.898 [0.140] (4.558×10^7)
0.05	0.762 [0.294] (3.484×10^7)	0.925 [0.160] (3.092×10^7)	0.989 [0.034] (2.136×10^7)	0.982 [0.061] (2.636×10^7)	**0.996 [0.015]** (2.81×10^7)	0.979 [0.034] (3.641×10^7)
0.1	0.992 [0.027] (1.699×10^7)	0.988 [0.033] (2.021×10^7)	0.993 [0.025] (1.931×10^7)	0.994 [0.017] (2.384×10^7)	0.990 [0.030] (2.913×10^7)	0.978 [0.033] (4.089×10^7)
0.2	0.958 [0.053] (3.927×10^7)	0.977 [0.038] (3.546×10^7)	0.972 [0.045] (3.582×10^7)	0.967 [0.050] (3.96×10^7)	0.906 [0.066] (4.791×10^7)	0.839 [0.073] (5×10^7)

Table A5. Average fitness of the controllers evolved with the SSS algorithm with the best combination of parameters (*MutRate* = 0.05; *PopSize* = 200, see Table A4). Different levels of noise were applied. Data were obtained in 30 replications of the experiment. Data in square brackets indicate the standard deviation. Data in round brackets represent the average number of evaluation episodes performed. Bold values indicate the best results.

NoiseRange	Fitness
0.0	**0.996 [0.015]** (2.81×10^7)
0.01	0.838 [0.097] (4.963×10^7)
0.02	0.814 [0.097] (5×10^7)
0.03	0.831 [0.010] (4.86×10^7)
0.04	0.801 [0.087] (4.983×10^7)
0.05	0.804 [0.112] (5×10^7)
0.06	0.840 [0.100] (4.702×10^7)
0.07	0.839 [0.092] (5×10^7)
0.08	0.813 [0.100] (4.944×10^7)
0.09	0.830 [0.087] (5×10^7)
0.1	0.801 [0.087] (4.991×10^7)
0.15	0.817 [0.098] (4.923×10^7)
0.2	0.828 [0.091] (4.899×10^7)

Table A6. Average fitness of the controllers evolved with the SSSHC algorithm. Experiments were run with the best combination of parameters found for the SSS algorithm (*MutRate* = 0.05; *PopSize* = 200, see Table A4). Experiments were run both without noise and with the best amount of noise found for the SSS algorithm (*NoiseRange* = 0.06, see Table A5). These data were collected by repeating the experiment 30 times. Data in square brackets indicate the standard deviation. Data in round brackets represent the average number of evaluation episodes performed. Bold values indicate the best results.

NumLearnIters	No Noise	Noisy
1	**0.991 [0.028]** (2.458×10^7)	0.885 [0.067] (4.948×10^7)
2	0.985 [0.028] (2.081×10^7)	0.901 [0.075] (4.848×10^7)
5	0.974 [0.037] (2.032×10^7)	0.936 [0.046] (4.676×10^7)
10	0.969 [0.039] (2.572×10^7)	0.931 [0.076] (4.884×10^7)
20	0.963 [0.049] (2.409×10^7)	0.939 [0.056] (4.672×10^7)
50	0.948 [0.061] (2.658×10^7)	**0.950 [0.052]** (4.386×10^7)
100	0.932 [0.063] (4.163×10^7)	0.947 [0.048] (4.677×10^7)

Appendix A.2.2. Randomly Varying Initial States Condition

Table A7. Average fitness of the controllers evolved with the SSS algorithm on the Randomly Varying Initial States condition in 30 replications of the experiment. Data in square brackets indicate the standard deviation. Bold values indicate the best results.

MutRate/PopSize	10	20	50	100	200	500
0.01	0.648 [0.212]	0.568 [0.259]	0.636 [0.235]	0.631 [0.205]	0.577 [0.198]	0.625 [0.133]
0.02	0.760 [0.217]	0.777 [0.181]	0.777 [0.209]	0.766 [0.194]	0.740 [0.141]	0.685 [0.138]
0.05	0.862 [0.140]	0.878 [0.156]	**0.903 [0.076]**	0.838 [0.117]	0.834 [0.076]	0.772 [0.098]
0.1	0.825 [0.145]	0.787 [0.131]	0.812 [0.098]	0.810 [0.104]	0.775 [0.104]	0.672 [0.12]
0.2	0.484 [0.209]	0.539 [0.171]	0.488 [0.160]	0.391 [0.176]	0.317 [0.162]	0.171 [0.080]

Table A8. Average fitness of the controllers evolved with the SSSHC algorithm on the Randomly Varying Initial States condition. Experiments were run with the best combination of parameters found for the SSS algorithm ($MutRate = 0.05$; $PopSize = 50$, see Table A7). These data come from 30 replications of the experiment. Data in square brackets indicate the standard deviation. Bold values indicate the best results.

NumLearnIters	No Noise
1	0.913 [0.097]
2	**0.939 [0.066]**
5	0.931 [0.076]
10	0.898 [0.048]
20	0.931 [0.059]
50	0.914 [0.065]
100	0.901 [0.067]

Appendix A.3. Optimization Functions

Appendix A.3.1. Rastrigin

Table A9. Average fitness of the controllers evolved with the SSSHC algorithm in the optimization of the Rastrigin function. Experiments were run with the same parameters used for the SSS and HC algorithms ($MutRate = 0.05$; $PopSize = 20$). These data come from 20 replications of the experiment. Data in square brackets indicate the standard deviation. Bold values indicate the best results (lower is better).

NumLearnIters	No Noise
1	**1.524 [0.0]**
2	**1.524 [0.0]**
5	**1.524 [0.0]**
10	**1.524 [0.0]**
20	**1.524 [0.0]**
50	**1.524 [0.0]**
100	1.720 [0.308]
200	1.931 [0.432]
500	2.325 [0.498]
1000	2.746 [0.573]

Appendix A.3.2. Rosenbrock

Table A10. Average fitness of the controllers evolved with the SSSHC algorithm in the optimization of the Rosenbrock function. Experiments were run with the same parameters used for the SSS and HC algorithms ($MutRate = 0.05$; $PopSize = 20$). These data come from 20 replications of the experiment. Data in square brackets indicate the standard deviation. Bold values indicate the best results (lower is better).

NumLearnIters	No Noise
1	27.513 [22.790]
2	24.573 [23.697]
5	30.547 [25.865]
10	24.938 [20.817]
20	24.567 [20.549]
50	18.895 [11.900]
100	**16.119 [2.100]**
200	16.799 [2.741]
500	17.732 [2.814]
1000	18.166 [3.722]

Appendix A.3.3. Sphere

Table A11. Average fitness of the controllers evolved with the SSSHC algorithm in the optimization of the Sphere function. Experiments were run with the same parameters used for the SSS and HC algorithms ($MutRate = 0.05$; $PopSize = 20$). These data come from 20 replications of the experiment. Data in square brackets indicate the standard deviation. Bold values indicate the best results (lower is better).

NumLearnIters	No Noise
1	**0.008 [0.0]**
2	**0.008 [0.0]**
5	**0.008 [0.0]**
10	**0.008 [0.0]**
20	**0.008 [0.0]**
50	0.009 [0.001]
100	0.011 [0.003]
200	0.012 [0.003]
500	0.014 [0.003]
1000	0.015 [0.004]

Appendix A.4. Robot Foraging

Table A12. Average fitness of the controllers evolved with the SSSHC algorithm in the Robot Foraging task. Experiments were run with the same parameters used for the SSS and HC algorithms ($MutRate = 0.01$; $PopSize = 20$). In the "Noisy" condition, the amount of noise $NoiseRange$ was set to 0.05. These data come from 20 replications of the experiment. Data in square brackets indicate the standard deviation. Bold values indicate the best results.

NumLearnIters	No Noise	Noisy
1	**8.870 [0.305]**	8.170 [0.354]
2	8.820 [0.340]	**8.350 [0.460]**
5	8.600 [0.358]	8.160 [0.436]

Appendix A.5. Social Foraging

Table A13. Average fitness of the controllers evolved with the SSSHC algorithm in the Social Foraging problem. Experiments were run with the same parameters used for the SSS and HC algorithms (*MutRate* = 0.01; *PopSize* = 20). In the "Noisy" condition, the amount of noise *NoiseRange* was set to 0.05. These data come from 20 replications of the experiment. Data in square brackets indicate the standard deviation. Bold values indicate the best results.

NumLearnIters	No Noise	Noisy
1	**1.450 [0.218]**	0.550 [0.244]
2	1.380 [0.303]	0.550 [0.267]
5	1.340 [0.229]	**0.620 [0.166]**

References

1. Prabhu, S.G.; Kyberd, P.J.; Melis, W.J.; Wetherall, J.C. Does lifelong learning affect mobile robot evolution? In *Recent Advances in Soft Computing and Cybernetics*; Springer: Cham, Switzerland, 2021; pp. 125–138.
2. Sznajder, B.; Sabelis, M.W.; Egas, M. How adaptive learning affects evolution: Reviewing theory on the Baldwin effect. *Evol. Biol.* **2012**, *39*, 301–310. [CrossRef] [PubMed]
3. Ackley, D.; Littman, M. Interactions between learning and evolution. *Artif. Life II* **1991**, *10*, 487–509.
4. Bengio, Y.; Bengio, S.; Cloutier, J. Learning a Synaptic Learning Rule. In *International Joint Conference on Neural Networks (IJCNN)*; IEEE: Seattle, WA, USA, 1991; pp. 969–974.
5. Chalmers, D.J. The evolution of learning: An experiment in genetic connectionism. In *Connectionist Models*; Elsevier: Amsterdam, The Netherlands, 1991; pp. 81–90.
6. Fontanari, J.F.; Meir, R. The effect of learning on the evolution of asexual populations. *Complex Syst.* **1990**, *4*, 401–414.
7. Gruau, F.; Whitley, D. Adding learning to the cellular development of neural networks: Evolution and the Baldwin effect. *Evol. Comput.* **1993**, *1*, 213–233. [CrossRef]
8. Hinton, G.E.; Nowlan, S.J. How learning can guide evolution. *Complex Syst.* **1987**, *1*, 495–502.
9. Mery, F.; Kawecki, T.J. The effect of learning on experimental evolution of resource preference in Drosophila melanogaster. *Evolution* **2004**, *58*, 757–767.
10. Nolfi, S.; Parisi, D.; Elman, J.L. Learning and evolution in neural networks. *Adapt. Behav.* **1994**, *3*, 5–28. [CrossRef]
11. Nolfi, S.; Parisi, D. Learning to adapt to changing environments in evolving neural networks. *Adapt. Behav.* **1996**, *5*, 75–98. [CrossRef]
12. Nolfi, S.; Floreano, D. Learning and evolution. *Auton. Robot.* **1999**, *7*, 89–113. [CrossRef]
13. Risi, S.; Hughes, C.E.; Stanley, K.O. Evolving plastic neural networks with novelty search. *Adapt. Behav.* **2010**, *18*, 470–491. [CrossRef]
14. Risi, S.; Stanley, K.O. Indirectly encoding neural plasticity as a pattern of local rules. In *International Conference on Simulation of Adaptive Behavior*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 533–543.
15. Soltoggio, A.; Bullinaria, J.A.; Mattiussi, C.; Dürr, P.; Floreano, D. Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. In Proceedings of the 11th International Conference on the Simulation and Synthesis of Living Systems, Winchester, UK, 5–8 August 2008; MIT Press: Cambridge, MA, USA, 2008; pp. 569–576.
16. Suzuki, R.; Arita, T. A simple computational model of the evolution of a communicative trait and its phenotypic plasticity. *J. Theor. Biol.* **2013**, *330*, 37–44. [CrossRef]
17. Whitley, D.; Gordon, V.S.; Mathias, K. Lamarckian evolution, the Baldwin effect and function optimization. In Proceedings of the Parallel Problem Solving from Nature—PPSN III: International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature, Jerusalem, Israel, 9–14 October 1994; Proceedings 3. Springer: Berlin/Heidelberg, Germany, 1994; pp. 5–15.
18. Yamauchi, B.M.; Beer, R.D. Sequential behavior and learning in evolved dynamical neural networks. *Adapt. Behav.* **1994**, *2*, 219–246. [CrossRef]
19. Ancel, L.W. Undermining the Baldwin expediting effect: Does phenotypic plasticity accelerate evolution? *Theor. Popul. Biol.* **2000**, *58*, 307–319. [CrossRef]
20. Anderson, R.W. Learning and evolution: A quantitative genetics approach. *J. Theor. Biol.* **1995**, *175*, 89–101. [CrossRef]
21. Borenstein, E.; Meilijson, I.; Ruppin, E. The effect of phenotypic plasticity on evolution in multipeaked fitness landscapes. *J. Evol. Biol.* **2006**, *19*, 1555–1570. [CrossRef] [PubMed]
22. Dopazo, H.; Gordon, M.; Perazzo, R.; Risau-Gusman, S. A model for the interaction of learning and evolution. *Bull. Math. Biol.* **2001**, *63*, 117–134. [CrossRef] [PubMed]
23. Nagrani, N. *Nature vs. Nurture: Effects of Learning on Evolution*; University of Toronto: Toronto, ON, Canada, 2010.
24. Paenke, I.; Kawecki, T.J.; Sendhoff, B. The influence of learning on evolution: A mathematical framework. *Artif. Life* **2009**, *15*, 227–245. [CrossRef]

25. Paenke, I.; Sendhoff, B.; Rowe, J.; Fernando, C. On the adaptive disadvantage of Lamarckianism in rapidly changing environments. In Proceedings of the Advances in Artificial Life: 9th European Conference, ECAL 2007, Lisbon, Portugal, 10–14 September 2007; Proceedings 9. Springer: Berlin/Heidelberg, Germany, 2007; pp. 355–364.
26. Floreano, D.; Husbands, P.; Nolfi, S. Evolutionary robotics. In *Handbook of Robotics*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 1423–1451.
27. Arena, P.; Patané, L.; Vitanza, A. Autonomous learning of collaboration among robots. In Proceedings of the The 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, QLD, Australia, 10–15 June 2012; pp. 1–8.
28. Borenstein, E.; Ruppin, E. Enhancing autonomous agents evolution with learning by imitation. In Proceedings of the Second International Symposium on Imitation in Animals and Artifacts, Aberystwyth, UK, 7–11 April 2003.
29. Paenke, I.; Jin, Y.; Branke, J. Balancing population-and individual-level adaptation in changing environments. *Adapt. Behav.* **2009**, *17*, 153–174. [CrossRef]
30. Schembri, M.; Mirolli, M.; Baldassarre, G. Evolution and learning in an intrinsically motivated reinforcement learning robot. In Proceedings of the Advances in Artificial Life: 9th European Conference, ECAL 2007, Lisbon, Portugal, 10–14 September 2007; Proceedings 9. Springer: Berlin/Heidelberg, Germany, 2007; pp. 294–303.
31. Saito, N.; Ishihara, S.; Kaneko, K. Baldwin effect under multipeaked fitness landscapes: Phenotypic fluctuation accelerates evolutionary rate. *Phys. Rev. E* **2013**, *87*, 052701. [CrossRef]
32. DeWitt, T.J.; Sih, A.; Wilson, D.S. Costs and limits of phenotypic plasticity. *Trends Ecol. Evol.* **1998**, *13*, 77–81. [CrossRef]
33. Mayley, G. Landscapes, learning costs, and genetic assimilation. *Evol. Comput.* **1996**, *4*, 213–234. [CrossRef]
34. Mayley, G. Guiding or hiding: Explorations into the effects of learning on the rate of evolution. In Proceedings of the Fourth European Conference on Artificial Life, Brighton, UK, 28–31 July 1997; Bradford Books/MIT Press: Cambridge, MA, USA, 1997; Volume 97, pp. 135–144.
35. Paenke, I.; Kawecki, T.J.; Sendhoff, B. On the influence of lifetime learning on selection pressure. In Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems, Bloomington, IN, USA, 3–6 June 2006; MIT Press: Cambridge, MA, USA, 2006; pp. 500–506.
36. Paenke, I.; Sendhoff, B.; Kawecki, T.J. Influence of plasticity and learning on evolution under directional selection. *Am. Nat.* **2007**, *170*, E47–E58. [CrossRef] [PubMed]
37. Price, T.D.; Qvarnström, A.; Irwin, D.E. The role of phenotypic plasticity in driving genetic evolution. *Proc. R. Soc. Lond. Ser. B Biol. Sci.* **2003**, *270*, 1433–1440. [CrossRef] [PubMed]
38. Soltoggio, A.; Stanley, K.O.; Risi, S. Born to learn: The inspiration, progress, and future of evolved plastic artificial neural networks. *Neural Netw.* **2018**, *108*, 48–67. [CrossRef]
39. Suzuki, R.; Arita, T. The dynamic changes in roles of learning through the Baldwin effect. *Artif. Life* **2007**, *13*, 31–43. [CrossRef]
40. Wiles, J.; Watson, J.; Tonkes, B.; Deacon, T. Transient phenomena in learning and evolution: Genetic assimilation and genetic redistribution. *Artif. Life* **2005**, *11*, 177–188. [CrossRef]
41. Pagliuca, P.; Milano, N.; Nolfi, S. Maximizing adaptive power in neuroevolution. *PLoS ONE* **2018**, *13*, e0198788. [CrossRef]
42. Moscato, P.; Cotta, C.; Mendes, A. Memetic algorithms. In *Handbook of Approximation Algorithms and Metaheuristics*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2002.
43. Neri, F.; Cotta, C.; Moscato, P. *Handbook of Memetic Algorithms*; Springer: Berlin/Heidelberg, Germany, 2011; Volume 379.
44. Nguyen, P.T.H.; Sudholt, D. Memetic algorithms beat evolutionary algorithms on the class of hurdle problems. In Proceedings of the Genetic and Evolutionary Computation Conference, Kyoto, Japan, 15–19 July 2018; pp. 1071–1078.
45. Prügel-Bennett, A. When a genetic algorithm outperforms hill-climbing. *Theor. Comput. Sci.* **2004**, *320*, 135–153. [CrossRef]
46. Holland, J.H. Adaptation in natural and artificial systems, univ. of mich. press. *Ann Arbor* **1975**, *7*, 390–401.
47. Rödl, V.; Tovey, C. Multiple optima in local search. *J. Algorithms* **1987**, *8*, 250–259. [CrossRef]
48. Wang, H.; Wang, D.; Yang, S. A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems. *Soft Comput.* **2009**, *13*, 763–780. [CrossRef]
49. Juels, A.; Wattenberg, M. Stochastic hillclimbing as a baseline method for evaluating genetic algorithms. In Proceedings of the Advances in Neural Information Processing Systems, Denver, CO, USA, 27–30 November 1995; Volume 8.
50. de Lamarck, J.B.d.M. *Philosophie Zoologique ou Exposition des Considérations Relatives à L'histoire Naturelle des Animaux*; Savy: Paris, France, 1873; Volume 1.
51. Grefenstette, J.J. Lamarckian learning in multi-agent environments. In Proceedings of the ICGA, San Diego, CA, USA, 13–16 July 1991; pp. 303–310.
52. Ku, K.W.C.; Mak, M.W.; Siu, W.C. Adding learning to cellular genetic algorithms for training recurrent neural networks. *IEEE Trans. Neural Netw.* **1999**, *10*, 239–252. [CrossRef] [PubMed]
53. Ku, K.W.C.; Mak, M.W.; Siu, W.C. A study of the Lamarckian evolution of recurrent neural networks. *IEEE Trans. Evol. Comput.* **2000**, *4*, 31–42. [CrossRef]
54. Prellberg, J.; Kramer, O. Lamarckian evolution of convolutional neural networks. In Proceedings of the Parallel Problem Solving from Nature—PPSN XV: 15th International Conference, Coimbra, Portugal, 8–12 September 2018; Proceedings, Part II 15. Springer: Berlin/Heidelberg, Germany, 2018; pp. 424–435.
55. Gupta, A.; Savarese, S.; Ganguli, S.; Li, F.-F. Embodied intelligence via learning and evolution. *Nat. Commun.* **2021**, *12*, 5721. [CrossRef]

56. Jelisavcic, M.; Kiesel, R.; Glette, K.; Haasdijk, E.; Eiben, A. Analysis of lamarckian evolution in morphologically evolving robots. In Proceedings of the Artificial Life Conference, Lyon, France, 4–8 September 2017; pp. 214–221.
57. Jelisavcic, M.; Glette, K.; Haasdijk, E.; Eiben, A. Lamarckian evolution of simulated modular robots. *Front. Robot. AI* **2019**, *6*, 9. [CrossRef]
58. Harada, K.; Iba, H. Lamarckian Co-design of Soft Robots via Transfer Learning. In Proceedings of the Genetic and Evolutionary Computation Conference, Melbourne, VIC, Australia, 14–18 July 2024; pp. 832–840.
59. Luo, J.; Miras, K.; Tomczak, J.; Eiben, A.E. Enhancing robot evolution through Lamarckian principles. *Sci. Rep.* **2023**, *13*, 21109. [CrossRef]
60. Jiaojiao, M.; Gulyás, L.; Botzheim, J. Comparing Lamarckian and Baldwinian Approaches in Memetic Optimization. In *International Conference on Computational Collective Intelligence*; Springer: Cham, Switzerland, 2023; pp. 521–533.
61. Pagliuca, P.; Nolfi, S. Integrating learning by experience and demonstration in autonomous robots. *Adapt. Behav.* **2015**, *23*, 300–314. [CrossRef]
62. Wieland, A.P. Evolving neural network controllers for unstable systems. In Proceedings of the IJCNN-91-Seattle International Joint Conference on Neural Networks, Seattle, WA, USA, 8–12 July 1991; Volume 2, pp. 667–673.
63. Gomez, F.; Schmidhuber, J.; Miikkulainen, R. Accelerated Neural Evolution through Cooperatively Coevolved Synapses. *J. Mach. Learn. Res.* **2008**, *9*, 937–965.
64. Igel, C. Neuroevolution for reinforcement learning using evolution strategies. In Proceedings of the The 2003 Congress on Evolutionary Computation, 2003. CEC'03, Canberra, ACT, Australia, 8–12 December 2003; Volume 4, pp. 2588–2595.
65. Stanley, K.O.; Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evol. Comput.* **2002**, *10*, 99–127. [CrossRef]
66. Wierstra, D.; Schaul, T.; Glasmachers, T.; Sun, Y.; Peters, J.; Schmidhuber, J. Natural evolution strategies. *J. Mach. Learn. Res.* **2014**, *15*, 949–980.
67. Rastrigin, L.A. Systems of extremal control. In *Theoretical foundations of engineering cybernetics series*; Nauka: Moscow, Russia, 1974.
68. Rosenbrock, H. An automatic method for finding the greatest or least value of a function. *Comput. J.* **1960**, *3*, 175–184. [CrossRef]
69. Dhawan, D.; Singh, R. Performance Evaluation of Nature Inspired Meta-Heuristic Algorithms using Rosenbrock, Rastrigin and Sphere Test Function for Optimization. *Int. J. Recent Technol. Eng* **2019**, *8*, 1157–1163.
70. Shi, Y.; Eberhart, R.C. Empirical study of particle swarm optimization. In Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), Washington, DC, USA, 6–9 July 1999; Volume 3, pp. 1945–1950.
71. Pagliuca, P.; Inglese, D.; Vitanza, A. Measuring emergent behaviors in a mixed competitive-cooperative environment. *Int. J. Comput. Inf. Syst. Ind. Manag. Appl.* **2023**, *15*, 69–86.
72. Pagliuca, P.; Vitanza, A. N-Mates Evaluation: A New Method to Improve the Performance of Genetic Algorithms in Heterogeneous Multi-Agent Systems. In Proceedings of the 24th Workshop from Object Agents (WOA23), Rome, Italy, 6–8 November 2023; Volume 3579, pp. 123–137.
73. Pagliuca, P.; Vitanza, A. The role of n in the n-mates evaluation method: A quantitative analysis. In *2024 Artificial Life Conference (ALIFE 2024)*; MIT Press: Cambridge, MA, USA, 2024; pp. 812–814.
74. Eryoldaş, Y.; Durmuşoglu, A. A literature survey on offline automatic algorithm configuration. *Appl. Sci.* **2022**, *12*, 6316. [CrossRef]
75. Liu, S.; Tang, K.; Lei, Y.; Yao, X. On performance estimation in automatic algorithm configuration. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020; Volume 34, pp. 2384–2391.
76. Milano, N.; Pagliuca, P.; Nolfi, S. Robustness, evolvability and phenotypic complexity: Insights from evolving digital circuits. *Evol. Intell.* **2019**, *12*, 83–95. [CrossRef]
77. Miller, J.F. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, FL, USA, 13–17 July 1999; Volume 2, pp. 1135–1142.
78. Massera, G.; Ferrauto, T.; Gigliotta, O.; Nolfi, S. Farsa: An open software tool for embodied cognitive science. In Proceedings of the Artificial Life Conference, Taormina, Italy, 2–6 September 2013; pp. 538–545.
79. Massera, G.; Ferrauto, T.; Gigliotta, O.; Nolfi, S. Designing adaptive humanoid robots through the FARSA open-source framework. *Adapt. Behav.* **2014**, *22*, 255–265. [CrossRef]
80. Aldana-Franco, F.; Montes-González, F.; Nolfi, S. The improvement of signal communication for a foraging task using evolutionary robotics. *J. Appl. Res. Technol.* **2024**, *22*, 90–101. [CrossRef]
81. Carvalho, J.T.; Nolfi, S. Behavioural plasticity in evolving robots. *Theory Biosci.* **2016**, *135*, 201–216. [CrossRef]
82. Palacios-Leyva, R.; Aldana-Franco, F.; Lara-Guzmán, B.; Montes-González, F. The impact of population composition for cooperation emergence in evolutionary robotics. *Int. J. Comb. Optim. Probl. Inform.* **2017**, *8*, 20–32.
83. Simione, L.; Nolfi, S. Long-term progress and behavior complexification in competitive coevolution. *Artif. Life* **2020**, *26*, 409–430. [CrossRef]
84. Koza, J. On the programming of computers by means of natural selection. *Genet. Program.* **1992**, *1*, 1–836.
85. Pagliuca, P.; Nolfi, S. Robust optimization through neuroevolution. *PLoS ONE* **2019**, *14*, e0213193. [CrossRef] [PubMed]
86. Carvalho, J.T.; Nolfi, S. The role of morphological variation in evolutionary robotics: Maximizing performance and robustness. *Evol. Comput.* **2024**, *32*, 125–142. [CrossRef] [PubMed]

87. Triebold, C.; Yaman, A. Evolving Generalist Controllers to Handle a Wide Range of Morphological Variations. In Proceedings of the Genetic and Evolutionary Computation Conference, Melbourne, VIC, Australia, 14–18 July 2024; pp. 1137–1145.
88. Demirol, D.; Oztemiz, F.; KARCI, A. Performance comparison of physics based meta-heuristic optimization algorithms. In Proceedings of the 2018 International Conference on Artificial Intelligence and Data Processing (IDAP), Malatya, Turkey, 28–30 September 2018; pp. 1–5.
89. Suganthan, P.N.; Hansen, N.; Liang, J.J.; Deb, K.; Chen, Y.P.; Auger, A.; Tiwari, S. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. *KanGAL Rep.* **2005**, *2005005*, 2005.
90. Winfield, A.F. Towards an engineering science of robot foraging. In *Distributed Autonomous Robotic Systems 8*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 185–192.
91. Mondada, F.; Bonani, M.; Raemy, X.; Pugh, J.; Cianci, C.; Klaptocz, A.; Magnenat, S.; Zufferey, J.C.; Floreano, D.; Martinoli, A. The e-puck, a robot designed for education in engineering. In Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, Castelo Branco, Portugal, 7 May 2009; IPCB: Instituto Politécnico de Castelo Branco: Castelo Branco, Portugal, 2009; Volume 1, pp. 59–65.
92. Whitley, D.; Kauth, J. GENITOR: A different genetic algorithm. In Proceedings of the 4th Rocky Mountain Conference on Artificial Intelligence, Denver, CO, USA, 13–15 June 1988.
93. Rechenberg, I. Evolutionsstrategie: Optimierung Technischer Systeme Nach Prinzipien der Biologischen Evolution. Frommann–Holzboog: Stuttgart-Bad Cannstatt, Germany, 1973; pp. 1–170.
94. Schwefel, H.P. *Numerische Optimierung von Computer-Modellen Mittels der Evolutionsstrategie: Mit Einer Vergleichenden Einführung in Die Hill-Climbing-und Zufallsstrategie*; Springer: Berlin/Heidelberg, Germany, 1977; Volume 1.
95. Kimura, M. *The Neutral Theory of Molecular Evolution*; Cambridge University Press: Cambridge, UK, 1983.
96. Wagner, A. Robustness, evolvability, and neutrality. *FEBS Lett.* **2005**, *579*, 1772–1778. [CrossRef]
97. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
98. Song, S.; Miller, K.D.; Abbott, L.F. Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nat. Neurosci.* **2000**, *3*, 919–926. [CrossRef]
99. Vitanza, A.; Patané, L.; Arena, P. Spiking neural controllers in multi-agent competitive systems for adaptive targeted motor learning. *J. Frankl. Inst.* **2015**, *352*, 3122–3143. [CrossRef]
100. Blynel, J.; Floreano, D. Exploring the T-maze: Evolving learning-like robot behaviors using CTRNNs. In *Workshops on Applications of Evolutionary Computation*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 593–604.
101. Kuo, K.C.; Chan, K.Y. Stable pushing in narrow passage environment using a modified hybrid A* algorithm. *J. Intell. Manuf.* **2024**, 1–14. [CrossRef]
102. Nolfi, S. Evolving non-trivial behaviors on real robots: A garbage collecting robot. *Robot. Auton. Syst.* **1997**, *22*, 187–198. [CrossRef]
103. Dorigo, M. SWARM-BOT: An experiment in swarm robotics. In Proceedings of the 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005, Pasadena, CA, USA, 8–10 June 2005; pp. 192–200.
104. Pagliuca, P.; Vitanza, A. Self-organized Aggregation in Group of Robots with OpenAI-ES. In *International Conference on Soft Computing and Pattern Recognition*; Springer: Cham, Switzerland, 2022; pp. 770–780.
105. Pagliuca, P.; Vitanza, A. Evolving aggregation behaviors in swarms from an evolutionary algorithms point of view. In *Applications of Artificial Intelligence and Neural Systems to Data Science*; Springer: Singapore, 2023; pp. 317–328.
106. Pagliuca, P.; Vitanza, A. Enhancing Aggregation in Locomotor Multi-Agent Systems: A Theoretical Framework. In Proceedings of the 25th Edition of the Workshop from Object to Agents (WOA24), Forte di Bard, Italy, 8–10 July 2024; Volume 3735, pp. 42–57.
107. Hansen, N.; Ostermeier, A. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.* **2001**, *9*, 159–195. [CrossRef] [PubMed]
108. Schaul, T.; Glasmachers, T.; Schmidhuber, J. High dimensions and heavy tails for natural evolution strategies. In Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, Dublin, Ireland, 12–16 July 2011; pp. 845–852.
109. Glasmachers, T.; Schaul, T.; Yi, S.; Wierstra, D.; Schmidhuber, J. Exponential natural evolution strategies. In Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, Portland, OR, USA, 7–11 July 2010; pp. 393–400.
110. Pagliuca, P.; Milano, N.; Nolfi, S. Efficacy of modern neuro-evolutionary strategies for continuous control optimization. *Front. Robot. AI* **2020**, *7*, 98. [CrossRef] [PubMed]
111. Pagliuca, P.; Nolfi, S. The dynamic of body and brain co-evolution. *Adapt. Behav.* **2022**, *30*, 245–255. [CrossRef]
112. Salimans, T.; Ho, J.; Chen, X.; Sidor, S.; Sutskever, I. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv* **2017**, arXiv:1703.03864.
113. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
114. Lillicrap, T. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.