Journal of Artificial Intelligence and Autonomous Intelligence
Received 15-07-2025; Accepted 19-08-2025; Published 26-08-2025

Discovering a Single Neural Network Controller for Multiple Tasks with Evolutionary Algorithms

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

Corresponding Author: Paolo Pagliuca

Copyright © 2025 Paolo Pagliuca. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract

Multi-Objective Optimization is a prominent research area, in which approaches for the simultaneous solution of multiple objectives are proposed. The possibility to discover a set of parameters optimizing all the goals can be achieved only if the considered problems are rather trivial, while compromise solutions are generally discovered. Things become even more complex when the set of parameters is used in opposite, and potentially conflicting, ways. In this work, we compared some state-of-the-art Evolutionary Algorithms with regard to the optimization of different conflicting objectives, by highlighting strengths and weaknesses of the different approaches. In particular, we considered four benchmark problems — 4-bit parity, double-pole balancing, grid navigation and test function optimization — to be solved simultaneously. Our investigation identifies the algorithms leading to a better optimization. In particular, three algorithms emerge as the most suitable methods for dealing with the considered scenario. Notably, a relatively simple strategy is not significantly inferior to a more sophisticated one. Moreover, we illustrate the solutions discovered by the different methods to address the benchmark problems.

Keywords: Benchmarking, Evolutionary Algorithms, Multi-Objective Optimization, Neural Networks.

1. INTRODUCTION

Multi-Objective Optimization (MOO) [1], is a fascinating research field in which the ultimate goal consists of discovering solutions fulfilling multiple, often conflicting, objectives simultaneously [1]. Since the task is extremely complex or even impossible, compromise solutions are taken into account, which are referred to as Pareto-optimal [2]. By definition, a Pareto-optimal solution is "a set of 'non-inferior' solutions in the objective space defining a boundary beyond which none of the objectives can be improved without sacrificing at least one of the other objectives" [3]. An illustration is provided in FIGURE 1.

Several techniques have been introduced to address MOO problems, such as Evolutionary Algorithms (EAs) [4–6], Genetic Algorithms (GAs) [7–9], or swarm intelligence approaches [10–12].

Citation: Paolo Pagliuca. Discovering a Single Neural Network Controller for Multiple Tasks with Evolutionary Algorithms. Journal of Artificial Intelligence and Autonomous Intelligence. Research. 2025;2(2):20.

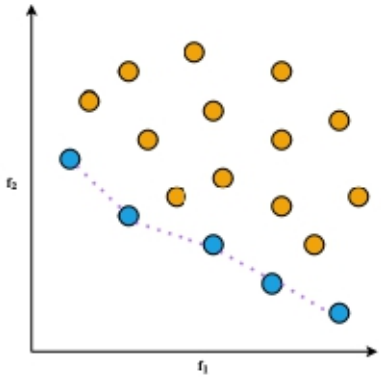

Figure 1: Explanation of Pareto optimality in a two-objective optimization (e.g., functions f_1 and f_2) scenario: non-dominated solutions (blue circles) are Pareto-optimal and lie on a theoretical line (purple dotted curve) termed as Pareto front. Remaining solutions (orange circles) are dominated by those belonging to the Pareto front.

For example, the authors in [13], applied GAs to a MOO scenario entailing a Supply Chain Network (SCN) design problem. Their analysis indicates the feasibility of the approach in the context of a Turkish company producing plastic products. In [14, 15], the OpenAI Evolutionary Strategy (OpenAI-ES) [16], has been used to cope with a MOO problem requiring a swarm of AntBullet robots [17], to simultaneously locomote and aggregate: the authors show that robots develop a good locomotion capability, while aggregation is obtained very rarely. This underscores the difficulty of optimizing both objectives at the same time. Specific algorithms have been developed to address MOO with regard to test function optimization, such as Nondominated Sorting Genetic Algorithm II (NSGA-II) [18], or Vector Evaluated Genetic Algorithm (VEGA) [19]. In particular, NSGA-II [18] enables to discover optimized solutions for test functions defined in [19, 20], by exploiting non-dominated sorting, diversity-preservation and elitism combined with typical mutation and crossover operators from GAs. VEGA [19], represents a variant of a classic GA tailored for MOO scenarios. Specifically, the solution evaluation produces a vector of fitness values, one for each objective, and selection is made by choosing non-dominated solutions according to Pareto optimality. To this end, the whole population is split into sub-populations (one for each objective) from which the best individuals are selected. Finally, recombination and mutation of selected solutions allow to generate the new population and ensure diversity.

An interesting field related to MOO involves the discovery of effective controllers to manage different problems belonging to the same class. For example, the authors in [21], propose an approach called Shared Modular Policy (SMP) in which a global policy is used to control different modular neural networks. In particular, each module is responsible for the control of a single actuator based on current sensory readings and shared information in the form of current reward function. Experiments performed on robot locomotors characterized by diverse morphologies (e.g., Halfcheetah, Hopper, Humanoid, Walker2D) demonstrate the validity of the approach and a good generalization capability on new morphologies. Another worthwhile example can be found in [22], in which the authors propose an approach to discover a single controller for a class of dynamical systems. Specifically, through in-context learning [23], the authors demonstrate that an effective controller

for a specific numerical problem can be immediately applied to a different problem within the same class, with no need for fine-tuning.

This work is halfway between MOO and the evolution of a single controller for multiple problems. In more detail, we present a comparative study of eight EAs — Generational Genetic Algorithm (GGA) [24], Hill-Climbing Algorithm (HC) [25], a variant of HC called HC*, OpenAI-ES [16], Separable Natural Evolution Strategies (sNES) [26], Stochastic Steady State [27], Stochastic Steady State with Hill-Climbing (SSSHC) [28], and an adaptive variant of SSSHC called SSSHCa — that must discover a single neural network controller capable to minimize simultaneously four benchmark problems: (i) 4-bit parity, (ii) double-pole balancing, (iii) grid navigation and (iv) test function optimization. Although using methods specifically tailored for MOO scenarios like NSGA-II or VEGA could be rather intuitive, we decided instead to put emphasis on the usage of general and relatively simple EAs [29], to assess how effective they are at sampling the search space and discovering optimized solutions for the MOO scenario. Furthermore, existing studies show that OpenAI-ES proved capable of dealing with MOO problems involving group of robots [15, 30, 31]. Similarly, GGA has been used to effectively solve robotic MOO tasks [32, 33]. In addition, the HC algorithm demonstrated its efficacy in optimizing multiple test functions [34, 35]. Our results clearly demonstrate that OpenAI-ES, SSSHC and SSSHCa outperform the other methods thanks to their propensity to reduce the size of weights, which is pivotal in the considered scenario. Furthermore, detailed analysis of the performance on the individual benchmark problems reveal that all the algorithms focus mainly on the minimization of test functions, which allows to quickly optimize performance. Interestingly, SSSHC and SSSHCa best OpenAI-ES with respect to grid navigation and test function optimization, while the opposite is true for 4-bit parity and double-pole balancing. This implies that relatively simple strategies can perform similarly to more sophisticated algorithms.

This research provides the following key contributions:

- a novel MOO scenario is presented, which entails benchmark problems like 4-bit parity, double-pole balancing, grid navigation and test function optimization;
- a comparison of eight EAs on the defined MOO scenario is proposed, including both classic methods (e.g., GGA, HC and SSS) and relatively novel and sophisticated algorithms (e.g., OpenAI-ES and sNES);
- two novel variants of existing methods, called HC* and SSSHCa, are introduced;
- OpenAI-ES, SSSHC and SSSHCa emerge as more effective than other methods at dealing with the novel MOO scenario, which indicate that relatively simple algorithms like SSSHC and SSSHCa are not inferior to a rather sophisticated method such as OpenAI-ES;
- SSSHC and SSSHCa outperform OpenAI-ES with respect to grid navigation and test function optimization, while the opposite is true for 4-bit parity and double-pole balancing;
- OpenAI-ES, SSSHC and SSSHCa manage to reduce the size of weights/parameters, which is paramount to discover effective solutions in the considered domain. In particular, OpenAI-ES exploits historic information to effectively sample the search space, while SSSHC and SSSHCa make use of single-gene mutations to discover truly adaptive modifications.

The paper starts with a thorough description of the benchmark problems, the considered EAs, the neural network controller and the experimental settings in Section 2. Then, the outcomes are illustrated in detail in Section 3, and a exhaustive discussion is provided in Section 4. Lastly, Section 5 reports the main findings and potential future research ideas.

2. MATERIALS AND METHODS

2.1 Problems

2.1.1 4-bit parity

The first problem is the 4-bit parity task, which consists in calculating the number of 1-bits in the input string and returning a value checking if the sum of 1-bits is even (output is 1) or odd (output is 0). The problem is depicted in FIGURE 2, and is a commonly used task in the evolutionary computation literature [36–38].

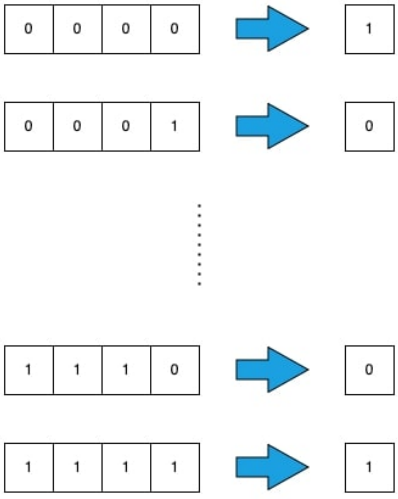

Figure 2: 4-bit parity problem: given a 4-bit input string, the objective is to return a 1/0 value when the number of 1-bits is even/odd. It is worth noting that the parity value for a sequence made of 0-bits only is 1.

Specifically, our evaluation is made by considering all the possible 4-bit input strings (i.e., $2^4 = 16$) and verifying the number of correct parity values returned. To this end, we design the following fitness function (Eq. 1).

$$F_{parity} = \sum_{s=1}^{N_{strings}} |O_s - \hat{O}_s| \tag{1}$$

In Eq. 1, the symbol O_s is the expected parity output for the input string s, $\hat{O}_s$ indicates the output returned by the controller and $N_{strings}$ represents the number of considered 4-bit input strings ($N_{strings} = 16$).

2.1.2 Double-pole balancing

The second problem is the double-pole balancing task [39], i.e. a widespread benchmark to assess an algorithm performance as demonstrated by the large body of literature in the field [27, 40–44]. The task involves the presence of two poles placed on the top of a wheeled mobile cart, which has to move properly to avoid poles falling (see FIGURE 3). TABLE 1, reports the parameters characterizing the task. The validity range of the cart position x is $[-2.4, 2.4]m$ (i.e., the length of the track), while the validity range of the pole angles θ_1 and θ_2 is $[-36, 36]°$.

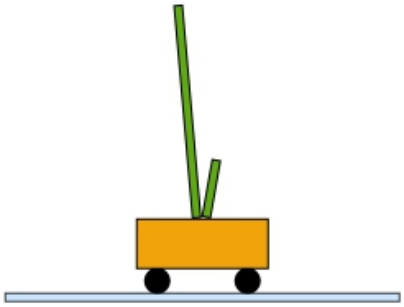

Figure 3: Double-pole balancing problem: a wheeled mobile cart (orange rectangle) has to move on a horizontal surface (light blue rectangle) in order to keep two poles (green rectangles) upright.

Table 1: Parameters characterizing the double-pole balancing problem. The track extremes are placed at -2.4 m and 2.4 m, respectively. We denote the absence of information with the symbol "-".

Parameter	Length	Mass
Track	4.8 m	-
Cart	-	1.0 kg
Long pole	1.0 m	0.5 kg
Short pole	0.1 m	0.05 kg

The objective function for this task is formulated as in Eq. 2.

$$F_{dpole} = \frac{1}{N_{trials}} \sum_{i=1}^{N_{trials}} N_{steps} - n_{steps}(i) \tag{2}$$

Symbol N_{steps} denotes the length of a trial ($N_{steps} = 1000$), $n_{steps}(i)$ in Eq. 2 represents the number of steps both poles are kept upright by the cart during the trial i, whereas N_{trials} indicates the number of trials. An episode is prematurely ended in two cases:

- the cart goes out of the track ($|x| > 2.4m$);
- at least one of the poles is falling ($|\theta_j| > 36°$, with $j = 1, 2$).

In this work, we focus on the "Fixed Initial States condition" version of the problem [27, 28], in which the evaluation of possible solutions is averaged over 8 trials ($N_{trials} = 8$). In more detail, at each trial the state variables ξ_i (with $i = 1 \dots 6$), which encode respectively position and velocity of the cart and the two poles, are initialized according to TABLE 2.

Table 2: Initialization of the state variables $\xi_1 \dots \xi_6$ in each trial of the double-pole balancing problem. Variables are defined as follows: $\xi_1 = x$; $\xi_2 = \dot{x}$; $\xi_3 = \theta_1$; $\xi_4 = \dot{\theta}_1$; $\xi_5 = \theta_2$; $\xi_6 = \dot{\theta}_2$.

Trial	ξ_1	ξ_2	ξ_3	ξ_4	ξ_5	ξ_6
1	-1.944	0	0	0	0	0
2	1.944	0	0	0	0	0
3	0	-1.215	0	0	0	0
4	0	1.215	0	0	0	0
5	0	0	-0.10472	0	0	0
6	0	0	0.10472	0	0	0
7	0	0	0	-0.135088	0	0
8	0	0	0	0.135088	0	0

Further details on the system dynamics can be found in [27, 28, 39].

2.1.3 Grid navigation

The third problem is a grid navigation task in which an agent has to reach the central location (target) of a square grid starting from one of the four corners (see FIGURE 4). The grid has size $L_G \times L_G$, with $L_G = 501$. Similarly to the afore-described problems, grid navigation represents a largely used benchmark to evaluate genetic and evolutionary algorithms [45–47].

The objective function for this problem is expressed as in Eq. 3.

$$F_{grid} = \frac{1}{N_{trials}} \sum_{i=1}^{N_{trials}} d_{ta}(i) \tag{3}$$

In Eq. 3, the symbol $d_{ta}(i)$ indicates the final agent-target distance at the trial i and N_{trials} counts the number of trials. Since there are four possible starting locations (i.e., the corners of the grid),

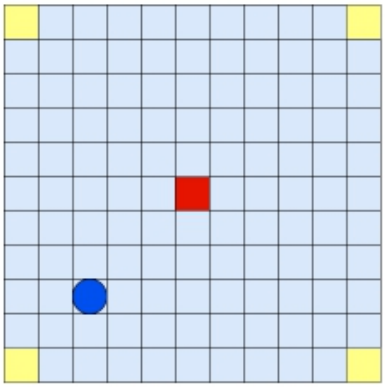

Figure 4: Grid navigation problem: an agent (red circle) has to navigate in a grid world. The agent starts from one of four possible initial locations (yellow squares) and its goal is to reach the target location (red square). The agent can move left, up, right or bottom.

we set $N_{trials} = 4$. The target-agent distance $d_{ta}(i)$ is computed by counting the minimum number of cells the agent has to visit in order to arrive at the target, and is defined in Eq. 4.

$$d_{ta}(i) = |t_x(i) - a_x(i)| + |t_y(i) - a_y(i)| \tag{4}$$

The symbols $(t_x(i), t_y(i))$ and $(a_x(i), a_y(i))$ denote, respectively, the location of the target and the agent during the trial i.

The agent can move one cell either horizontally (i.e., left and right) or vertically (i.e., up and down) within the grid. A trial prematurely stops if the action of the agent causes its exit from the grid.

2.1.4 Test function optimization

The last problem is test function optimization, which constitutes a common benchmark for evaluating optimization methods [18, 48–50]. In particular, we consider the Ackley, Griewank, Rastrigin, Rosenbrock and Sphere functions. Eqs. 5 - 9 provide the definition of the considered functions, in which the input sequence is identified with the symbol v and its length with the symbol n.

$$F_{ackley} = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} {v_i}^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi v_i)\right) + 20 + \exp(1) \tag{5}$$

$$F_{griewank} = 1 + \frac{1}{4000}\sum_{i=1}^{n} {v_i}^2 - \prod_{i=1}^{n}\cos\left(\frac{v_i}{\sqrt{i}}\right) \tag{6}$$

$$F_{rastrigin} = \sum_{i=1}^{n} ({v_i}^2 - 10\cos(2\pi v_i) + 10) \tag{7}$$

$$F_{rosenbrock} = \sum_{i=1}^{n-1} (100({v_{i+1}} - {v_i}^2)^2 + (v_i - 1)^2) \tag{8}$$

$$F_{sphere} = \sum_{i=1}^{n} {v_i}^2 \tag{9}$$

The fitness function for test function optimization is defined as the sum of the single functions according to Eq. 10.

$$F_{funct} = F_{ackley} + F_{griewank} + F_{rastrigin} + F_{rosenbrock} + F_{sphere} \tag{10}$$

2.2 Performance

To assess the overall performance of the discovered solutions with respect to the considered MOO scenario, we adopt the fitness function described in Eqs. 11 - 12:

$$\min F \tag{11}$$

$$F = w_{parity}F_{parity} + w_{dpole}F_{dpole} + w_{grid}F_{grid} + w_{funct}F_{funct} \tag{12}$$

As concerns Eq. 12, we use coefficients equal to 1 for each objective (i.e., $w_{parity} = w_{dpole} = w_{grid} = w_{funct} = 1$). Despite the simplicity of the approach, our design choice alleviates from the burden of choosing tailored values, which requires expertise and may lead to different evolutionary paths.

2.3 Evolutionary Algorithms

The EAs employed in this work are described in the next sections. A comprehensive schematic is provided in FIGURE 5. Further details about the different algorithms can be found in [16, 24–28].

We emphasize that the term “solution” represents a set of floating-point values (also called “genes”) directly encoding the parameters (i.e., biases and connection weights) of the controller that must be evolved.

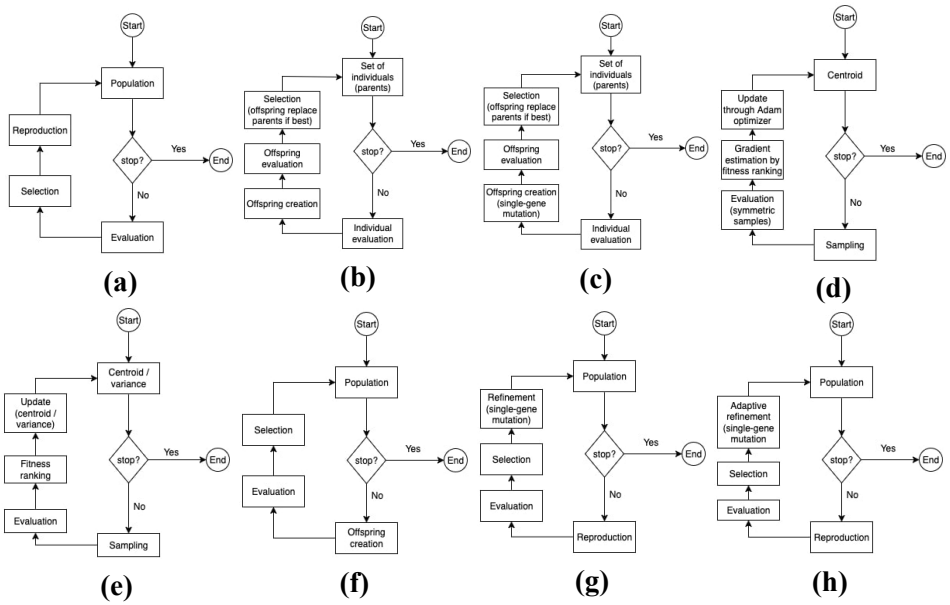

Figure 5: Schematic of the considered EAs. (a) GGA; (b) HC; (c) HC*; (d) OpenAI-ES; (e) sNES; (f) SSS; (g) SSSHC; (h) SSSHCa.

2.3.1 GGA

The Generational Genetic Algorithm (GGA) [24] is a pioneering method to evolve solutions capable of adapting to dynamic environments [51, 52]. Moreover, it has been largely used in collective domains like Swarm Robotics [53–56] or Multi-Agent Systems (MASs) [57–61], as well as for problems involving single agents [62–65].

The algorithm works according to the schematic provided in FIGURE 5-(a) and evolves a population of candidate solutions. Each solution undergoes an evaluation process returning a fitness value rating its effectiveness. After the entire population has been evaluated, the best N_r solutions are selected for reproduction, each one generating N_o offspring through mutation and asexual crossover operators. Moreover, selected solutions are retained in the population through elitism. This iterative process aims to enhance the efficacy of solutions during evolution.

2.3.2 HC and HC*

The Hill-Climbing (HC) algorithm [25] is an optimization technique working through local search refinements. Differently from classic population methods like GGA or SSS, HC considers a pool of individuals that are evolved independently. Specifically, at each iteration, individuals generate one offspring each and both are evaluated. The selection process takes into account each pair $(individual, offspring)$ and retains the best performing solution within the pool of individuals. For each individual, offspring is generated through mutation or asexual crossover. The HC algorithm is shown in FIGURE 5-(b).

HC* is a variant of HC in which the offspring creation is made through single-gene mutations (see the schematic in FIGURE 5-(c)). Although the mechanism theoretically ensures the discovery of truly adaptive modifications, there is no guarantee that solutions are progressively improved, especially when only a gene at a time is mutated.

2.3.3 OpenAI-ES

The OpenAI Evolutionary Strategy (OpenAI-ES) [16] is a relatively novel technique that achieved competitive results in problems like Atari games [16], classic control [66, 67], competitive co-evolution [68], robot locomotion [16, 66, 69] and swarm robotics [30, 31, 70]. Differently from a traditional EA, OpenAI-ES evolves a single solution termed "centroid". In particular, at each iteration of the evolutionary process, $N_{samples}$ samples are extracted from a Gaussian distribution and a set of solutions are derived according to Eq. 13.

$$s \rightarrow \begin{cases} z^{+} = c + \sigma \times s \\ z^{-} = c - \sigma \times s \end{cases} \tag{13}$$

In Eq. 13, the symbol s represents the generic sample, c denotes the centroid, z^{+} and z^{-} indicate the solutions that undergo evaluation, while σ is the mutation rate.

Once the pool of solutions has been evaluated, a fitness ranking is used to estimate the gradient of the expected fitness function. Finally, the centroid is updated through Adam [71], a widespread optimizer keeping historical information to drive the search process towards solutions that are more likely to be effective. A schematic of OpenAI-ES is shown in FIGURE 5-(d). Differently from existing works, here OpenAI-ES does not exploit weight decay [72], since reducing the size of weights represents a clear advantage in the considered MOO scenario.

2.3.4 sNES

The Separable Natural Evolution Strategies (sNES) algorithm [26] is a variant of the Exponential Natural Evolution Strategies (xNES) [44] that does not compute any covariance matrix. Instead, sNES calculates the variance for each gene separately, hence allowing the method to better scale with increasing sizes of the search space [66]. In particular, sNES has been successfully employed in test function optimization [26], double-pole balancing [44, 73], car racing games [73], swarm robotics [73] and robot locomotion [66]. The schematic of sNES is reported in FIGURE 5-(e). A detailed description of sNES can be found in [26, 44].

2.3.5 SSS

The Stochastic Steady State (SSS) [27] is an EA that found applications in problems like double-pole balancing [27, 28, 73], swarm robotics [73, 74], car racing games [73] and robot locomotion [75]. The algorithm differs from GGA with respect to the reproduction and selection process: at each iteration, each solution in the population (parent) generates one offspring through mutation and

asexual crossover. Then, both parents and offspring are evaluated and the best $PopSize$ solutions are retained and form the next population. This iterative process allows SSS to improve performance during evolution. FIGURE 5-(f) provides a schematic of the operation of SSS.

2.3.6 SSSHC and SSSHCa

The Stochastic Steady State with Hill-Climbing (SSSHC) [28] is a memetic algorithm [76–78] combining SSS with a Hill-Climbing algorithm [25] seeking to further refine the solution performance. SSSHC proves successful with regard to problems such as 5-bit parity, double-pole balancing and test function optimization [28, 49]. Specifically, for the experiments reported here, we used the variant introduced in [49].

The operation of SSSHC follows the one of SSS, but a refinement phase is added after selection: each selected solution undergoes N_{refs} iterations (with $N_{refs} = 5$ as in [49]) in which a single-gene mutation is applied and the modified solution is retained only if its performance improves. A description of SSSHC is illustrated in FIGURE 5-(g).

Moreover, we designed an adaptive variant of SSSHC, called SSSHCa, in which the parameter N_{refs} adapts during evolution based on the number of non-negative single-gene mutations discovered during the refinement process (see FIGURE 5-(h)). Specifically, we compute the percentage of non-negative mutations α_m according to Eq. 14.

$$\alpha_m = \frac{N_m^+}{N_m} \tag{14}$$

where the symbol N_m^+ and N_m indicate, respectively, the number of non-negative mutations and the total number of mutations performed during the refinement process.

The current value of α_m is compared with the previous stored value (denoted with $\bar{\alpha}_m$), and N_{refs} is updated according to Eq. 15. To avoid numerical issues, we constrained the parameter N_{refs} in the range $[1, 25]$. As for SSSHC, N_{refs} has been initially set to 5.

$$\Delta N_{refs} = \begin{cases} +1 & \text{if } \alpha_m > \bar{\alpha}_m \\ 0 & \text{if } \alpha_m = \bar{\alpha}_m \\ -1 & \text{if } \alpha_m < \bar{\alpha}_m \end{cases} \tag{15}$$

2.4 Controller

As a single controller for the considered problems, we used a recurrent neural network [79] formed by 4 inputs, an internal layer of 10 neurons, and 1 output. Bias is applied to internal and output neurons, which activate based on the *tanh* function. The information the input and output neurons encode depending on the specific problem is listed here below (see also TABLE 3).

- 4-bit parity: the inputs represent the bits of the string that has to be checked, while the output is the parity value associated to the string;
- double-pole balancing: the inputs encode normalized values for the cart position (x) and pole angles (θ_1 and θ_2), and an alert signal that activates if either the cart approaches track edges ($|x| > 2m$), or the poles are almost falling ($|\theta_j| > 30°$, with $j = 1, 2$). The output represents the force that will be exerted to the cart;
- grid navigation: the inputs are the normalized locations of the agent ((a_x, a_y)) and the target ((t_x, t_y)), whereas the output indicates the motion direction (i.e., left, up, right or bottom).

Regarding test function optimization, the connection weights of the neural network constitute the input vector v used to compute the functions in Eq. 5 - 9. Therefore, not only does the controller have to optimize simultaneously four problems exhibiting different properties, but it also has to address them in potentially conflicting ways. In fact, the latter problem requires to minimize the size of connection weights, which could prevent the discovery of effective solutions for the other problems.

Table 3: Encoding of the input (I_1, I_2, I_3, I_4) and output (O_1) neurons used for 4-bit parity, double-pole balancing and grid navigation problems. Symbols are defined as follows: concerning 4-bit parity, bit_j (with $j = 1 \dots 4$) states for the generic bit of the input string, whereas n_{1-bits} indicates the network output used to check parity. As regards double-pole balancing, x refers to the cart position, θ_1 and θ_2 denote the angle of the long and short poles, respectively, and $alert$ is a flag indicating whether the trial might prematurely be stopped because either the cart is going out of the track ($|x| > 2$) or the pole angles are above 30°, while $force$ is the force applied to the cart that determines its motion. Lastly, with respect to grid navigation, the symbols (a_x, a_y) and (t_x, t_y) represent, respectively, the position of the agent and of the target locations, L_G is the grid size and dir is the direction of the agent in the grid (with $dir \in [left, up, right, down]$).

Problem	I_1	I_2	I_3	I_4	O_1
4-bit parity	bit_1	bit_2	bit_3	bit_4	n_{1-bits}
Double-pole balancing	$\frac{x}{4.8}$	$\frac{\theta_1}{0.52}$	$\frac{\theta_2}{0.52}$	$alert$	$force$
Grid navigation	$\frac{a_x}{L_G - 1}$	$\frac{a_y}{L_G - 1}$	$\frac{t_x}{L_G - 1}$	$\frac{t_y}{L_G - 1}$	dir

2.5 Simulator and Experimental Settings

We employed he Framework for Autonomous Robotics Simulation and Analysis (FARSA) [80] tool, since it found applications in similar experimental studies [27, 28, 49]. Moreover, it has been successfully used also in robotic scenarios [65, 74, 81, 82].

The considered EAs have been compared by running 30 replications of the experiments, and evolution lasts 10^9 evaluation steps. TABLE 4 reports a list of all parameters.

Table 4: List of parameter settings used for the different algorithms.

Parameter	Symbol	GGA	HC	HC*	OpenAI-ES	sNES	SSS	SSSHC	SSSHCa
Number of replications	N_{reps}	30							
Number of evaluation steps	N_{evals}	10^9							
Population size	$PopSize$	100	50		1		50		
Number of reproducing solutions	N_r	10	-						
Number of offspring generated by each solution	N_o	$\frac{PopSize}{N_r} - 1$	-						
Elitism	$Elitism$	Yes	-						
Mutation rate	$MutRate$	0.05			0.02	Adaptive based on variance	0.05		
Crossover rate	$CrossRate$	0.1	-				0.1		
Learning rate	$LearnRate$	-			0.01	-			
Number of samples	$N_{samples}$	-			20	$4 + \lfloor 3 \times \ln(n) \rfloor$	-		
Number of refinement iterations	N_{refs}	-						5	Adaptive based on α_m
Weight range	w_r	$[-5.0, 5.0]$							

3. RESULTS

The performance comparison is shown in TABLE 5, FIGURE 6 and FIGURE 7. Clearly, OpenAI-ES, SSSHC and SSSHCa strongly outperform GGA, HC, HC*, sNES and SSS (Kruskal-Wallis H test, $p < 10^{-6}$, see TABLE 6), which indicates their capability to discover more effective solutions for the considered MOO scenario. As can be seen in FIGURE 6, OpenAI-ES quickly reduces the fitness in the first 10^8 evaluation steps and then almost stabilizes after $2{\times}10^8$ evaluation steps. HC*, SSSHC and SSSHCa have a slower convergence and stabilize after $4{\times}10^8$ evaluation steps, although they reach different performance levels. The sNES algorithm immediately reduces performance, but does not manage to further improve during evolution. Conversely, the fitness curves of GGA, HC and SSS show a slower drop than other methods, which prevents them from further optimization.

Notably, OpenAI-ES, SSSHC and SSSHCa have similar performance (Mann-Whitney U test with Bonferroni correction, $p > 0.05$ for each pair of comparisons, see also TABLE 6). Therefore, relatively simple strategies like SSSHC and SSSHCa are not inferior to a more sophisticated algorithm like OpenAI-ES, which exploits historical information to channel the search in the space of possible solutions. By looking at the outcomes reported in FIGURE 7, OpenAI-ES manages to find solutions with performance below 1000 (see bottom outliers in FIGURE 7), while SSSHC and SSSHCa do not reach similar performance levels.

HC* strongly outperforms HC (Mann-Whitney U test, $p < 10^{-6}$, see also TABLE 6). This implies that single-gene mutations allow the former method to improve performance, since these modifications are truly adaptive. Conversely, the exploration process of HC through random mutations over multiple genes or asexual crossover prevents the method from discovering effective solutions, particularly with respect to test function optimization (see TABLE 7).

Table 5: Fitness analysis of the different algorithms. Data is the average of 30 replications of the experiments. Best performance is reported in bold.

GGA	HC	HC*	OpenAI-ES	sNES	SSS	SSSHC	SSSHCa
2915.823 [206.120]	4342.209 [221.425]	1715.914 [40.453]	**1291.433 [268.231]**	1788.938 [28.329]	2581.040 [117.146]	1423.723 [57.654]	1420.432 [45.443]

We corroborate our results by delving into the fitness achieved by the different algorithms with regard to the specific problems. As highlighted in FIGURE 8 and TABLE 7, OpenAI-ES achieves the best results in the 4-bit parity and double-pole balancing problems, while SSSHCa bests other algorithms with respect to grid navigation and test function optimization. In particular, OpenAI-ES,

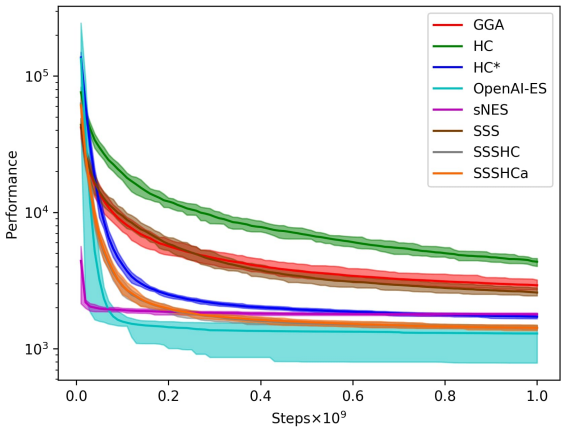

Figure 6: Performance of GGA, HC, HC*, OpenAI-ES, sNES, SSS, SSSHC and SSSHCa during evolution. The shaded areas are bounded in the range $[Q_1, Q_3]$ (first and third quartiles of data). We use the logarithmic scale on the y-axis to improve readability. Fitness is averaged over 30 replications.

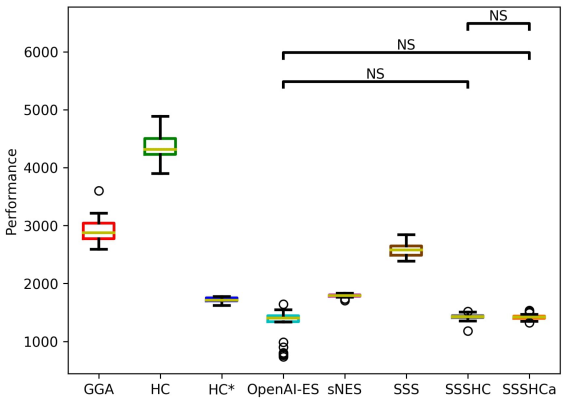

Figure 7: Final performance achieved by the different methods (see TABLE 5). Boxes are bounded in the range $[Q_1, Q_3]$, with the whiskers extending to data within $1.5\times(Q_3-Q_1)$. Medians are indicated with yellow lines. The notation *NS* indicates that the fitness values of the two considered methods do not statistically differ (Mann-Whitney U test with Bonferroni correction, $p > 0.05$, see also TABLE 6).

Table 6: Statistical comparison between the considered methods according to the Mann-Whitney U test with Bonferroni correction, with significant differences indicated in bold. Table is symmetrical with respect to the main diagonal. The symbol "-" marks the absence of the corresponding entry. Data is the average of 30 replications of the experiments.

	GGA	HC	HC*	OpenAI-ES	sNES	SSS	SSSHC	SSSHCa
GGA	-	**$< 10^{-6}$**	**$< 10^{-6}$**	**$< 10^{-6}$**	**$< 10^{-6}$**	**$< 10^{-6}$**	**$< 10^{-6}$**	**$< 10^{-6}$**
HC		-	**$< 10^{-6}$**	**$< 10^{-6}$**	**$< 10^{-6}$**	**$< 10^{-6}$**	**$< 10^{-6}$**	**$< 10^{-6}$**
HC*			-	**$< 10^{-6}$**	**$< 10^{-6}$**	**$< 10^{-6}$**	**$< 10^{-6}$**	**$< 10^{-6}$**
OpenAI-ES				-	**$< 10^{-6}$**	**$< 10^{-6}$**	0.061	0.186
sNES					-	**$< 10^{-6}$**	**$< 10^{-6}$**	**$< 10^{-6}$**
SSS						-	**$< 10^{-6}$**	**$< 10^{-6}$**
SSSHC							-	0.232
SSSHCa								-

SSSHC and SSSHCa succeed in the minimization of test functions, which allows to quickly reduce fitness (see also FIGURE 6). Instead, GGA, HC, HC*, sNES and SSS fail in finding similar solutions. Interestingly, OpenAI-ES is the only algorithm that manages to discover improved solutions in the double-pole balancing problem (Kruskal-Wallis H test, $p < 10^{-6}$, see also FIGURE 8 and TABLE 7).

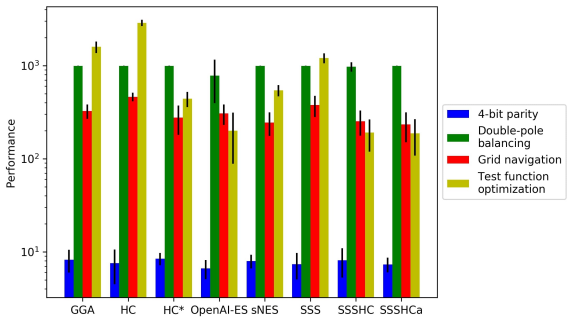

Figure 8: Analysis of the algorithm performance on the different problems. Black lines mark the standard deviations. We use the logarithmic scale on the y-axis to improve readability. Bars denote the average fitness from 30 replications.

Table 7: Performance analysis with regard to 4-bit parity, double-pole balancing, grid navigation and test function optimization. Bold values correspond to the best outcomes. Data is the average of 30 replications of the experiments.

Problem	GGA	HC	HC*	OpenAI-ES	sNES	SSS	SSSHC	SSSHCa
4-bit parity	8.233 [2.246]	7.533 [3.052]	8.433 [1.283]	**6.633 [1.538]**	7.967 [1.329]	7.367 [2.331]	7.800 [1.078]	7.333 [1.300]
Double-pole balancing	994.654 [2.199]	994.275 [2.566]	990.846 [6.199]	**778.279 [384.148]**	993.963 [2.625]	993.708 [2.394]	971.204 [112.980]	993.163 [2.949]
Grid navigation	325.167 [57.660]	462.075 [48.844]	276.250 [96.566]	306.142 [75.842]	245.008 [69.218]	377.092 [97.762]	253.058 [76.517]	**232.800 [82.173]**
Test function optimization	1587.770 [223.944]	2878.325 [219.006]	440.385 [81.073]	200.379 [111.980]	542.000 [74.207]	1202.873 [147.151]	191.361 [72.429]	**187.136 [79.009]**

Lastly, we investigate the performance of the different methods with regard to Ackley, Griewank, Rastrigin, Rosenbrock and Sphere functions, as illustrated in FIGURE 9 and TABLE 8. Notably,

OpenAI-ES is more effective than other algorithms in optimizing the Rosenbrock function (Kruskal-Wallis H test, $p < 10^{-6}$), SSS outperforms the others with regard to Rastrigin function (Kruskal-Wallis H test, $p < 10^{-6}$), SSSHC reaches best performance on the Sphere function (Kruskal-Wallis H test, $p < 10^{-6}$), whereas SSSHCa bests other algorithms with respect to Ackley and Griewank functions (Kruskal-Wallis H test, $p < 10^{-6}$). The outcomes reported in TABLE 7 and TABLE 8 underscore the clear advantage of OpenAI-ES, SSSHC and SSSHCa over other methods. HC* and sNES obtain relatively good results in test function optimization, although they are inferior to OpenAI-ES, SSSHC and SSSHCa. GGA, HC and SSS achieve poor performance in this context, particularly concerning the Rosenbrock function (see TABLE 8).

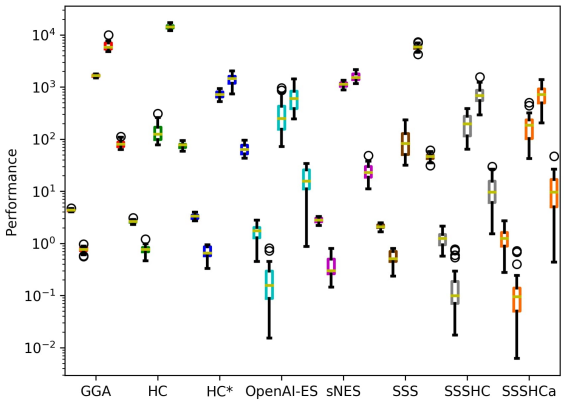

Figure 9: Algorithm performance on the Ackley, Griewank, Rastrigin, Rosenbrock and Sphere test functions. Boxes are bounded in the range $[Q_1, Q_3]$, with the whiskers extending to data within $1.5 \times (Q_3 - Q_1)$. Medians are indicated with yellow lines. We use the logarithmic scale on the y-axis to improve readability. Data represents the average fitness from 30 replications.

Table 8: Performance analysis with regard to the Ackley, Griewank, Rastrigin, Rosenbrock and Sphere test functions. Bold values correspond to the best outcomes. Data is the average of 30 replications of the experiments.

Test function	GGA	HC	HC*	OpenAI-ES	sNES	SSS	SSSHC	SSSHCa
Ackley	4.331 [0.179]	2.652 [0.151]	3.275 [0.304]	1.673 [0.586]	2.900 [0.227]	2.107 [0.184]	1.261 [0.434]	**1.254 [0.531]**
Griewank	0.765 [0.094]	0.761 [0.147]	0.696 [0.171]	0.217 [0.193]	0.447 [0.185]	0.546 [0.146]	0.181 [0.201]	**0.142 [0.167]**
Rastrigin	1643.956 [66.687]	138.704 [52.489]	720.609 [100.050]	332.113 [245.317]	1108.064 [99.170]	**97.339 [53.305]**	202.919 [96.824]	193.670 [106.812]
Rosenbrock	6207.401 [1112.739]	14173.956 [1083.805]	1411.610 [322.264]	**650.228 [317.946]**	1618.943 [212.284]	5868.017 [702.571]	741.408 [295.443]	728.661 [305.485]
Sphere	82.393 [12.567]	75.548 [7.719]	65.735 [15.735]	17.665 [9.097]	28.012 [7.952]	46.356 [6.654]	**11.038 [7.100]**	11.953 [9.570]

Overall, the reported analysis clearly reveals a superior performance of OpenAI-ES, SSSHC and SSSHCa over GGA, HC, HC*, sNES and SSS. This is related to their capability to decrease weights, as shown in TABLE 9. In particular, the former algorithms evolve controllers with a limited weight size, which is paramount for the test function optimization. Nevertheless, this tendency prevents the algorithms from finding effective solutions with respect to the other problems, particularly double-pole balancing. This is especially true for SSSHC and SSSHCa (see FIGURE 8 and TABLE 7). By examining the controllers evolved by the different methods, we discovered a positive correlation between performance and weight size (Spearman correlation, $\rho = 0.234$, significant at $p < 0.001$), which further emphasizes the importance of reducing weight size for the discovery of effective

solutions. Interestingly, sNES and SSS have similar weight sizes, but their performance is notably different. This can be explained by considering that sNES computes the variance for each gene, which allows the method to identify the modifications that could improve performance. Conversely, SSS does not adopt similar mechanisms. An analogous consideration can be done with regard to HC and HC*: the usage of single-gene mutations allows the latter method to discover truly adaptive modifications, whereas the former algorithm suffers from the potentially disruptive effect of multiple-gene mutations and gets stuck in local minima solutions.

Table 9: Controller weight size analysis. Data has been computed by considering the absolute value for each gene and is the average of 30 replications of the experiments.

GGA	HC	HC*	OpenAI-ES	sNES	SSS	SSSHC	SSSHCa
0.583 [0.040]	0.428 [0.041]	0.466 [0.083]	0.164 [0.071]	0.265 [0.040]	0.290 [0.042]	0.091 [0.051]	0.096 [0.068]

Regarding SSSHCa, we measured how the number of refinement iterations N_{refs} varies throughout evolution based on the percentage of non-negative mutations α_m found during the refinement process. As can be expected, parameter α_m decreases as the algorithm improves performance (FIGURE 10, top), because the possibility to find further adaptive mutations is low. Instead, the number of refinement iterations N_{refs} tends to increase during evolution (FIGURE 10, bottom). This is related to the update rule defined in Eq. 15, which fosters a great variability of the parameter N_{refs} even with low values of variable α_m. Future studies could delve into different, and more effective, techniques to adapt the number of refinement iterations N_{refs}.

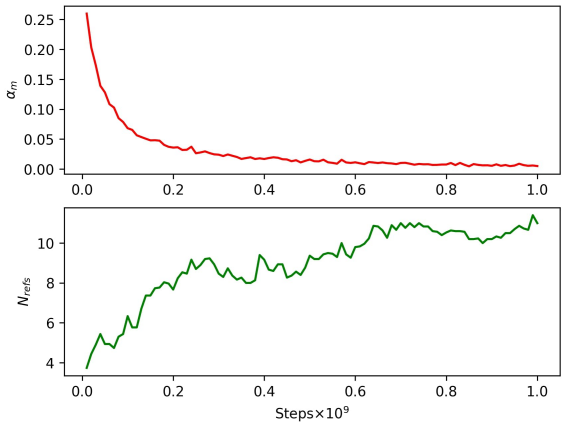

Figure 10: Analysis of the variation of parameter N_{refs} (bottom figure) depending on the rate of non-negative mutations α_m (top figure) throughout the evolution with SSSHCa. Data is collected from 30 replications.

4. DISCUSSION

The presented results show that three algorithms emerge as more suitable options for the considered MOO scenario. A noteworthy aspect regards the different level of complexity between OpenAI-ES,

SSSHC and SSSHCa. In fact, the former method exploits mirrored sampling [83] and historical information to identify the search direction in the space of solutions. Conversely, SSSHC and SSSHCa are memetic algorithms that seek to refine selected solutions through single-gene mutations. Despite the simplicity, SSSHC and SSSHCa work well in this scenario. However, this approach presents benefits and drawbacks: on one hand, modifying one gene at a time eliminates the risk of disruptive effects, since only adaptive mutations are retained. On the other hand, the process does not exclude failures in the attempt of improving solution performance and is costly with regard to the number of evaluation steps.

OpenAI-ES has an intrinsic propensity to reduce the size of weights even when weight decay is not applied, as in the case of our experiments. This turns out to be crucial in the considered domain, as well as in other domains like swarm robotics [31] and robot locomotion [69]. Conversely, SSSHC and SSSHCa exploit single-gene mutations performed during refinement to progressively reduce the weight size.

HC* and sNES achieve relatively good results, although significantly inferior to OpenAI-ES, SSSHC and SSSHCa. Despite the usage of single-gene mutations, HC* does not manage to properly explore the search space. A similar consideration can be hold for sNES. This implies that the advantage of OpenAI-ES, SSSHC and SSSHCa is related to their overall operation, rather than the single techniques incorporated in their implementation. In particular, OpenAI-ES effectively combines symmetric sampling and historical information to understand the parameter modifications that could lead to enhanced performance. On the other hand, SSSHC and SSSHCa balance exploration (through asexual crossover and random mutations) and exploitation (refinement by means of single-gene mutations) to improve discovered solutions.

GGA, HC and SSS do not manage to achieve performance comparable to the other algorithms. Specifically, they fail in discovering effective solutions for test function optimization, particularly with respect to the Rosenbrock function. This can be explained by considering that GGA, HC and SSS seek to enhance performance through mutation and crossover operators, and do not have explicit techniques to reduce weights. As a consequence, they might require longer evolutionary processes in order to find more effective solutions.

Although the difference in weight size, HC* is remarkably more effective than HC. This reveals that a pure parameter reduction does not correspond to an optimized performance. Instead, single-gene mutations allow to identify those parameters that must be modified in order to enhance fitness and better deal with the MOO scenario.

Another relevant insight regards the types of solutions discovered by the considered EAs. Specifically, we observe that all the algorithms mainly focus on test function optimization, which allows to quickly enhance performance, and almost ignore the remaining problems. This is strongly related to the remarkable different magnitudes of the fitness values in the worst case (see TABLE 10): because test function optimization is three order of magnitude bigger than double-pole balancing and grid navigation, performance is minimized when the component F_{funct} is strongly reduced (see Eq. 12). As a consequence, the algorithms evolve solutions mostly addressing test function optimization and do not manage to improve performance with respect to the other objectives.

Overall, our results underscore a non-negligible relationship between problem and algorithm. In fact, although OpenAI-ES, SSS and SSSHC proved effective at optimizing double-pole balancing

Table 10: Worst fitness value that can be obtained in each considered problem.

4-bit parity	Double-pole balancing	Grid navigation	Test function optimization
16	1000	500	6450690.655

in previous studies [27, 28, 67], the definition of the performance measure for the considered MOO scenario guides EAs to different evolutionary paths, in which one of the objectives (test function optimization) dominates the others.

Given the strong dependency of the types of solutions found on the fitness definition, we run a preliminary investigation on the strategies discovered by HC*, OpenAI-ES, sNES, SSSHC and SSSHCa (i.e., the most successful algorithms in the MOO scenario) when the fitness functions defined in Eqs. 1, 2, 3 and 10 are normalized in the range $[0.0, 1.0]$ according to Eqs. 16 - 19.

$$\tilde{F}_{parity} = \frac{F_{parity}}{N_{strings}} \tag{16}$$

$$\tilde{F}_{dpole} = \frac{F_{dpole}}{N_{steps}} \tag{17}$$

$$\tilde{F}_{grid} = \frac{F_{grid}}{L_G - 1} \tag{18}$$

$$\tilde{F}_{funct} = \frac{F_{funct}}{10^6} \tag{19}$$

Concerning Eqs. 16 - 18, the normalization factors $N_{strings}$, N_{steps} and $L_G - 1$ ensure that the normalized fitness functions are bounded in the range $[0, 1]$. With regard to Eq. 19, the value 10^6 has been empirically determined by observing the non-normalized performance of solutions found during the early stages of the evolutionary process.

The results of this analysis are presented in TABLE 11. The fitness normalization removes the dominating objective (i.e., test function optimization) and enables a more balanced optimization. Nonetheless, there are differences in the way the considered EAs address the modified MOO scenario: OpenAI-ES succeeds in discovering effective solutions for test function optimization and is able to enhance its performance on the double-pole balancing. Conversely, the other algorithms excel at solving 4-bit parity and grid navigation, and manage to find good solutions to test function optimization. Notably, SSSHC and SSSHCa are signicantly better than HC* and sNES (Mann-Whitney U test with Bonferroni correction, $p < 0.001$) and also outperform OpenAI-ES, although there is no statistical difference (Mann-Whitney U test with Bonferroni correction, $p > 0.05$). FIGURE 11 displays how performance varies throughout the evolution.

Interestingly, the fitness normalization affects the weight size of the evolved controllers (see TABLE 12). In fact, in this case, the performance is negatively correlated with the size of weights

Table 11: Performance of HC*, OpenAI-ES, sNES, SSSHC and SSSHCa when the fitness of single tasks (defined in Eqs. 1, 2, 3 and 10) is normalized in the range $[0.0, 1.0]$. Data is averaged over 30 replications. Best outcomes are shown in bold.

Problem	HC*	OpenAI-ES	sNES	SSSHC	SSSHCa
4-bit parity	0.008 [0.021]	0.158 [0.171]	**0.0 [0.0]**	0.010 [0.056]	0.004 [0.016]
Double-pole balancing	0.883 [0.128]	**0.506 [0.379]**	0.969 [0.096]	0.790 [0.212]	0.801 [0.220]
Grid navigation	0.155 [0.077]	0.375 [0.310]	**0.135 [0.043]**	0.137 [0.016]	0.138 [0.018]
Test function optimization	0.103 [0.030]	**0.033 [0.027]**	0.073 [0.017]	0.099 [0.033]	0.095 [0.021]
Total	1.149 [0.166]	1.072 [0.435]	1.178 [0.041]	**1.037 [0.212]**	1.038 [0.215]

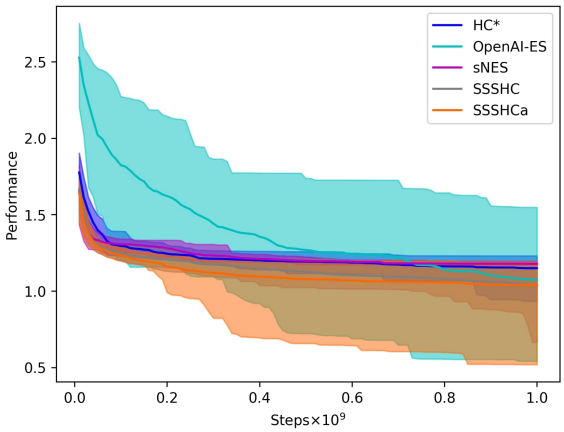

Figure 11: Performance of HC*, OpenAI-ES, sNES, SSSHC and SSSHCa during evolution with normalized fitness functions (see Eqs. 16 - 19). The shaded areas are bounded in the range $[Q_1, Q_3]$ (first and third quartiles of data). Fitness is averaged over 30 replications.

(Spearman correlation, $\rho = -0.294$, significant at $p < 0.001$). Therefore, the usage of normalized fitness functions drives the EAs towards the discovery of solutions characterized by larger weights than in the basic MOO scenario (Mann-Whitney U test, $p < 10^{-6}$ for each algorithm), which is paramount to balance optimization across multiple objectives.

Table 12: Weight size analysis of controllers evolved with normalized fitness. Data has been computed by considering the absolute value for each gene and is the average of 30 replications of the experiments.

HC*	OpenAI-ES	sNES	SSSHC	SSSHCa
1.470 [0.177]	1.167 [0.321]	1.356 [0.061]	1.410 [0.161]	1.419 [0.134]

5. CONCLUSIONS

Multi-Objective Optimization (MOO) concerns the identification of solutions capable of optimizing multiple conflicting objectives at the same time. Because the discovery of a global optimum is not trivial, if not impossible, Pareto-optimal solutions are typically taken into account. Evolutionary Algorithms (EAs) have demonstrated a great ability to deal with MOO scenarios, since they provide effective solutions without any prior knowledge about the considered domains. A partially related research field lies in the search for a single controller able to address different problems within the same class. The main advantage is the possibility of transferring acquired knowledge on a given problem to others, without further fine-tuning. The approach is similar to transfer learning [84], which proved valuable in Deep Learning (DL) applications [85–87].

This work delves into the definition of a novel MOO scenario and the analysis of how some state-of-the-art works address it. Specifically, we compared GGA, HC, HC*, OpenAI-ES, sNES, SSS, SSSHC and SSSHCa with respect to the evolution of a single neural network controller able to simultaneously optimize four benchmark problems: (i) 4-bit parity, (ii) double-pole balancing, (iii) grid navigation, and (iv) test function optimization. The definition of the scenario makes the overall problem challenging, since the multiple objectives conflict with each other. Furthermore, the controller is used differently depending on the specific problem: for example, the connection weights determine the action to be executed in the double-pole balancing or grid navigation problems, while they represent the input vector for the test functions to be optimized. Outcomes indicate that OpenAI-ES, SSSHC and SSSHCa best the other algorithms, since they are more effective at sampling the search space. In particular, OpenAI-ES exploits symmetric sampling and historical information, whereas SSSHC and SSSHCa benefit from single-gene mutations. Interestingly, relatively trivial algorithms like SSSHC and SSSHCa are not inferior to a modern and sophisticated method like OpenAI-ES in this context. Instead, SSSHC and SSSHCa are better than OpenAI-ES with respect to grid navigation and test function optimization, while the opposite is true for 4-bit parity and double-pole balancing. In addition, OpenAI-ES, SSSHC and SSSHCa manage to reduce weight size more efficiently than GGA, HC, HC*, sNES and SSS, a worthwhile feature in the considered domain. Lastly, a preliminary investigation on the usage of normalized fitness functions underscores that fitness normalization highly affects the capability of HC*, OpenAI-ES, sNES, SSSHC and SSSHCa (i.e., the best EAs in the MOO scenario) to balance optimization across multiple objectives, and leads to the discovery of solutions characterized by larger weight sizes.

As future research directions, we plan to incorporate algorithms specifically tailored for MOO domains, such as NSGA-II and VEGA, for future comparisons. This will provide a comprehensive analysis of the most suitable methods for MOO. Furthermore, modifications of the problem formulation in Eq. 12 will be object of future studies, particularly through the usage of different coefficient values for the different objectives. In addition, further experiments in which parameter settings are systematically varied will be performed, aiming to validate our analysis more thoroughly. Finally, future works could investigate different scenarios (e.g., robotics, classic control) in order to extend and, possibly, generalize the considerations reported here.

References

[1] Deb K, Sindhya K, Hakanen J. Multi-Objective Optimization. Decision sciences. CRC Press; 2016:145-184.

[2] Censor Y. Pareto Optimality in Multiobjective Problems. Appl Math Optim. 1977;4:41-59.

[3] Kara N, Köçken HG. An Approach for a Multi-Objective Capacitated Transportation Problem. Encyclopedia of data science and machine learning. IGI Global; 2023:2385-2399.

[4] Deb K. Multi-Objective Optimisation Using Evolutionary Algorithms: An Introduction. In: Wang L, Ng AH, Deb K, editors. Multi-objective evolutionary optimisation for product design and manufacturing. Springer; 2011:3-34.

[5] Fonseca CM, Fleming PJ. An Overview of Evolutionary Algorithms in Multiobjective Optimization. Evol Comput. 1995;3:1-16.

[6] Tan KC, Lee TH, Khor EF. Evolutionary Algorithms for Multi-Objective Optimization: Performance Assessments and Comparisons. Artif Intell Rev. 2002;17:251-290.

[7] Chafekar D, Xuan J, Rasheed K. Constrained Multi-Objective Optimization Using Steady State Genetic Algorithms. In: Genetic and evolutionary computation conference. Springer; 2003:813-824.

[8] Coello CA, Lamont GB, Van Veldhuizen DA. Evolutionary Algorithms for Solving Multi-Objective Problems. Springer; 2007.

[9] Konak A, Coit DW, Smith AE. Multi-Objective Optimization Using Genetic Algorithms: A Tutorial. Reliab Eng Syst Saf. 2006;91:992-1007.

[10] Alaya I, Solnon C, Ghedira K. Ant Colony Optimization for Multiobjective Optimization Problems. In: 19th IEEE International Conference on Tools With Artificial Intelligence (ICTAI 2007). IEEE. 2007;1:450-457.

[11] Janga Reddy M, Nagesh Kumar DN. An Efficient Multi-Objective Optimization Algorithm Based on Swarm Intelligence for Engineering Design. Eng Optim. 2007;39:49-68.

[12] Yasear SA, Ku-Mahamud KR. Review of the Multi-Objective Swarm Intelligence Optimization Algorithms. J Inf Commun Technol. 2021;20:171-211.

[13] Altiparmak F, Gen M, Lin L, Paksoy T. A Genetic Algorithm Approach for Multi-Objective Optimization of Supply Chain Networks. Comput Ind Eng. 2006;51:196-215.

[14] Pagliuca P, Vitanza A. Enhancing Aggregation in Locomotor Multi-Agent Systems: A Theoretical Framework. Proceedings of the 25th Edition of the Workshop From Object to Agents (WOA24). 2024;3735:42-57.

[15] Pagliuca P, Trivisano G, Vitanza A. How to Evolve Aggregation in Robotic Multi-Agent Systems. In: Proceedings of the 26th Edition of the Workshop From Object to Agents. (WOA25). 2025.

[16] Salimans T, Ho J, Chen X, Sidor S, Sutskever I. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. arXiv preprint arXiv: https://arxiv.org/pdf/1703.03864

[17] Coumans E, Bai Y. Pybullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning; 2016.

[18] Deb K, Pratap A, Agarwal S, Meyarivan T. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. IEEE Trans Evol Computat. 2002;6:182-197.

[19] Schaffer JD. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In: First International Conference on Genetic Algorithms and Their Applications. Psychology Press; 2014:93-100.

[20] Fonseca CM, Fleming PJ. Multiobjective Optimization and Multiple Constraint Handling With Evolutionary Algorithms. ii. Application Example. IEEE Trans Syst Man Cybern A. 1998;28:38-47.

[21] Huang W, Mordatch I, Pathak D. One Policy to Control Them All: Shared Modular Policies for Agent-Agnostic Control. In: International Conference on Machine Learning. PMLR. 2020:4455-4464.

[22] Busetto R, Breschi V, Forgione M, Piga D, Formentin S. One Controller to Rule Them All. Proceedings of the Machine Learning Research. 2025;1:14.

[23] Dong Q, Li L, Dai D, Ce Z, Ma J, Li R et al. A Survey on In-Context Learning. arXiv preprint arXiv: https://arxiv.org/pdf/2301.00234.

[24] Grefenstette JJ. Genetic Algorithms for Changing Environments. In: International Conference on Parallel Problem Solving From Nature (PPSN). 1992;2:137-144.

[25] Rödl V, Tovey C. Multiple Optima in Local Search. J Algor. 1987;8:250-259.

[26] 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. New York, USA: ACM; 2011:845-852.

[27] Pagliuca P, Milano N, Nolfi S. Maximizing Adaptive Power in Neuroevolution. PLOS One. 2018;13:e0198788.

[28] Pagliuca P. Learning and Evolution: Factors Influencing an Effective Combination. AI. 2024;5:2393-2432.

[29] Bäck T. Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford university press; 1996.

[30] Pagliuca P, Vitanza A. Self-Organized Aggregation in Group of Robots with OpenAI-ES. In: International Conference on Soft Computing and Pattern Recognition. Springer; 2022:770-780.

[31] Pagliuca P, Vitanza A. A Comparative Study of Evolutionary Strategies for Aggregation Tasks in Robot Swarms: Macro- And Micro-Level Behavioral Analysis. IEEE Access. 2025;13:72721-72735.

[32] JMouret JB, Doncieux S. Incremental Evolution of Animats' Behaviors as a Multi-Objective Optimization. In: International Conference on Simulation of Adaptive Behavior.Springer; 2008:210-219.

[33] Nolfi S, Parisi D. Evolving Non-trivial Behaviors on Real Robots: An Autonomous Robot That Picks up Objects. In: Congress of the Italian Association for Artificial Intelligence. Springer; 1995:243-254.

[34] Chicano F, Whitley D, Tinós R. Efficient Hill Climber for Multiobjective Evolutionary Computation in Combinatorial. Optimization. Proceedings, volume. 2016;9595:88.

[35] Díaz R, Suarez AR. A Study of the Capacity of the Stochastic Hill Climbing to Solve Multi-Objective Problems. In: Proceedings of the Third International Symposium on Adaptive Systems-Evolutionary Computation and Probabilistic Graphical Models. La Habana: Institute of Cybernetics, Mathematics and Physics; 2001:37-40.

[36] Liu D, Hohil ME, Smith SH. N-Bit Parity Neural Networks: New Solutions Based on Linear Programming. Neurocomputing. 2002;48:477-488.

[37] Miller JF. An Empirical Study of the Efficiency of Learning Boolean Functions Using a Cartesian Genetic Programming Approach. In: Genetic and Evolutionary Computation Conference. 1999;2:1135-1142.

[38] Youssef A, Majeed B, Ryan C. Optimizing Combinational Logic Circuits Using Grammatical Evolution. IEEE; 2021 3rd Novel Intelligent and Leading Emerging Sciences Conference (NILES). 2021:87-92.

[39] Wieland AP. Evolving Neural Network Controllers for Unstable Systems. In: International Joint Conference on Neural Networks (IJCNN). IEEE; 1991;2:667-673.

[40] Gomez F, Schmidhuber J, Miikkulainen R. Accelerated Neural Evolution Through Cooperatively Coevolved Synapses. J Mach Learn Res. 2008;9.

[41] Gruau F, Whitley D, Pyeatt L. A Comparison Between Cellular Encoding and Direct Encoding for Genetic Neural Networks. In: 1st Annual Conference on Genetic Programming; 1996:81-89.

[42] Igel C. Neuroevolution for Reinforcement Learning Using Evolution Strategies. In: The 2003 Congress on Evolutionary Computation. IEEE; 2003; 2:2588-2595.

[43] Stanley KO, Miikkulainen R. Evolving Neural Networks Through Augmenting Topologies. Evol Comput. 2002;10:99-127.

[44] Wierstra D, Schaul T, Glasmachers T, Sun Y, Peters J, et al,. Natural Evolution Strategies. J Mach Learn Res. 2014;15:949-980.

[45] Elshahed A, Bin Majahar Ali MK, Mohamed AS, Abdullah FA, Aun TL. Efficient Pathfinding on Grid Maps: Comparative Analysis of Classical Algorithms and Incremental Line Search. IEEE Access. 2025;13:98473-98484.

[46] Heng H, Rahiman W. Aco-Ga-Based Optimization to Enhance Global Path Planning for Autonomous Navigation in Grid Environments. IEEE Trans Evol Computat. 2025.

[47] Miglino O, Walker R. Genetic Redundancy in Evolving Populations of Simulated Robots. Artif Life. 2002;8:265-277.

[48] Adabor ES, Ackora-Prah J. A Genetic Algorithm on Optimization Test Functions. Int J Mod Eng Res. 2017;7:1-11.

[49] Pagliuca P. Analysis of the Exploration-Exploitation Dilemma in Neutral Problems with Evolutionary Algorithms. J Artif Intell Auton Intell. 2024;1:110-121.

[50] Song Y, Wang F, Chen X. An Improved Genetic Algorithm for Numerical Function Optimization. Appl Intell. 2019;49:1880-1902.

[51] Nolfi S, Parisi D. Learning to Adapt to Changing Environments in Evolving Neural Networks. Adapt Behav. 1996;5:75-98.

[52] Vavak F, Fogarty TC. Comparison of Steady State and Generational Genetic Algorithms for Use in Nonstationary Environments. In: IEEE International Conference on Evolutionary Computation. IEEE; 1996:192-195.

[53] Baldassarre G, Trianni V, Bonani M, Mondada F, Dorigo M, et al. Self-Organized Coordinated Motion in Groups of Physically Connected Robots. IEEE Trans Syst Man Cybern B Cybern (Cybernetics). 2007;37:224-239.

[54] Groß R, Dorigo M. Towards Group Transport by Swarms of Robots. Int J Bio Inspired Comput. 2009;1:1-13.

[55] Trianni V, Groß R, Labella TH, Şahin E, Dorigo M. Evolving Aggregation Behaviors in a Swarm of Robots. In: European Conference on Artificial Life. Springer; 2003:865-874.

[56] Trianni V, Nolfi S. Self-Organizing Sync in a Robotic Swarm: A Dynamical System View. IEEE Trans Evol Computat. 2009;13:722-741.

[57] 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.

[58] Pagliuca P, Vitanza A. N-Mates Evaluation: A New Method to Improve the Performance of Genetic Algorithms in Heterogeneous Multi-Agent Systems. Proceedings of the 24th Edition of the Workshop from Object to Agents (WOA23). 2023;3579:123-137.

[59] Pagliuca P, Vitanza A. The Role of N in the N-Mates Evaluation Method: A Quantitative Analysis. In: Artificial Life Conference (ALIFE 2024). MIT Press; 2024:812-814.

[60] Pagliuca P, Favia M, Livi S, Vitanza A. Conceptualizing Evolving Interdependence in Groups: Insights from the Analysis of Two-Agent Systems. In: 21st International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT); 2025.

[61] Pagliuca P, Favia M, Livi S, Vitanza A. Interdipendenza Nei Gruppi: Esperimenti Con Robot Sociali. Sistem Intell. 2025;37:335-355.

[62] Floreano D, Mondada F. Evolution of Homing Navigation in a Real Mobile Robot. IEEE Trans Syst Man Cybern B Cybern (Cybernetics). 1996;26:396-407.

[63] Nolfi S. Evolving Non-Trivial Behaviors on Real Robots: A Garbage Collecting Robot. Robot Auton Syst. 1997;22:187-198.

[64] Nolfi S, Marocco D. Evolving Robots Able to Integrate Sensory-Motor Information Over Time. Theor Biosci. 2001;120:287-310.

[65] Pagliuca P, Yuri Inglese DY. The Importance of Functionality Over Complexity: A Preliminary Study on Feed-Forward Neural Networks. In: Esposito A, Faundez-Zanuy M, Morabito FC, Pasero E, Cordasco G, editors. Advanced Neural Artificial Intelligence: Theories and Applications. Springer; 2025:447-458.

[66] Pagliuca P, Milano N, Nolfi S. Efficacy of Modern Neuro-Evolutionary Strategies for Continuous Control Optimization. Front Robot AI. 2020;7:98.

[67] Pagliuca P, Nolfi S, Vitanza A. Evorobotpy3: A Flexible and Easy-to-Use Simulation Tool for Evolutionary Robotics. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2025 Companion). New York, USA: ACM; 2025:155-158.

[68] Nolfi S, Pagliuca P. Global Progress in Competitive Co-Evolution: A Systematic Comparison of Alternative Methods. Front Robot AI. 2024;11:1470886.

[69] Pagliuca P, Nolfi S. The Dynamic of Body and Brain Co-Evolution. Adapt Behav. 2022;30:245-255.

[70] Rais Martínez J, Aznar Gregori F. Comparison of Evolutionary Strategies for Reinforcement Learning in a Swarm Aggregation Behaviour. In: 3rd International Conference on Machine Learning and Machine Intelligence; 2020:40-45.

[71] Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. 2014. arXiv Preprint arXiv: https://arxiv.org/pdf/1412.6980.

[72] Krogh A, Hertz J. A Simple Weight Decay Can Improve Generalization. Adv Neural Inf Process Syst. 1991;4.

[73] Pagliuca P, Nolfi S. Robust Optimization Through Neuroevolution. PLOS One. 2019;14:e0213193.

[74] Aldana-Franco F, Montes González F, Nolfi S. Evolutionary Utility of Emerging Communication Systems and Signal Complexity in Robotics. Int J Comb Optim Probl Inform. 2024;15:15-27.

[75] Respall VM, Nolfi S. Development of Multiple Behaviors in Evolving Robots. Robotics. 2020;10:1.

[76] Cotta C. Harnessing Memetic Algorithms: A Practical Guide. TOP. 2025;33:327-356.

[77] Moscato P, Cotta C, Mendes A. Memetic Algorithms. New Optim Tech Eng. 2004;141:53-85.

[78] Neri F, Cotta C, Moscato P. Handbook of Memetic Algorithms. Springer; 2011;379.

[79] Elman JL. Finding Structure in Time. Cogn Sci. 1990;14:179-211.

[80] Massera G, Ferrauto T, Gigliotta O, Nolfi S. FARSA: An Open Software Tool for Embodied Cognitive Science. In: Artificial Life Conference; 2013:538-545.

[81] Aldana-Franco F, Montes-González F, Nolfi S. Improvement of Signal Communication for a Foraging Task Using Evolutionary Robotics. J Appl Res Technol. 2024;22:90-101.

[82] Pagliuca P, Nolfi S. Integrating Learning by Experience and Demonstration in Autonomous Robots. Adapt Behav. 2015;23:300-314.

[83] Brockhoff D, Auger A, Hansen N, Arnold DV, Hohm T. Mirrored Sampling and Sequential Selection for Evolution Strategies. In: International Conference on Parallel Problem Solving From Nature. Springer; 2010:11-21.

[84] Torrey L, Shavlik J. Transfer Learning. In: Olivas ES, Guerrero JD, Martinez-Sober M, Magdalena-Benedito JR, Serrano López AJ, editors. Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques. IGI Global; 2010:242-264.

[85] Kandel I, Castelli M. Transfer Learning With Convolutional Neural Networks for Diabetic Retinopathy Image Classification: A Review. Appl Sci. 2020;10:2021.

[86] Pagliuca P, Zribi M, Tufo G, Pitolli F. The Trade-Off Between Efficiency, Sustainability and Explainability: A Comparative Study on the Quality Control of Laboratory Consumables. In: International Joint Conference on Neural Networks (IJCNN). IEEE; 2025.

[87] Zribi M, Pagliuca P, Pitolli F. A Computer Vision-Based Quality Assessment Technique for the Automatic Control of Consumables for Analytical Laboratories. Expert Syst Appl. 2024;256