# Out of Time: On the Constrains that Evolution in Hardware Faces When Evolving Modular Robots

Rodrigo Moreno(✉) and Andres Faiña

IT University of Copenhagen, Copenhagen, Denmark
rodr@itu.dk

**Abstract.** With the recent advances of modular robots and low-cost manipulators, the evolution of robots, including morphologies and controllers, has become possible to perform in a physical setup without using any simulators. In this scenario, the evolution cannot be parallelized and the wall time becomes a scarce resource that should be used wisely. This paper analyses different algorithms by using the wall time as a stopping criterion for evolution, and it takes into account that wall time depends on the evaluation time plus the time to assemble and disassemble robots before and after an evaluation. The experiments have been performed in simulation, but the evaluation and assembly time have been carefully modelled from previous hardware experiments. Results suggest that (i) genetic algorithms are severely penalized, (ii) genetic algorithms can be improved by performing several evaluations of controllers for each morphology, and that (iii) evolutionary strategies that can chain several evaluations of robots with close morphologies can outperform other evolutionary algorithms. This finding is not surprising, but to the best of our knowledge previous attempts to evolve modular robots in hardware have not employed evolutionary strategies.

**Keywords:** Modular robots · Evolutionary robotics · Morphological evolution · Evolution in hardware

## 1 Introduction

Autonomous robots help automate a lot of different tasks, but they must be designed in a suitable way to be successful. In order to obtain robot designs adapted to a specific task, several authors have proposed to evolve the morphology and controller of a robot at the same time [16,27]. These works perform the evolutionary process mostly in simulation as several morphological variations are evaluated in a short time. However, this advantage comes with an important drawback: the reality gap.

Due to simplified physics and bad modelled features, simulators are not capable of accurately simulating the real world. Therefore, evolution exploits these artifacts and produces robot designs that do not perform well in the real world. There are several approaches to reduce the effect of the reality gap. Some of


J. L. Jiménez Laredo et al. (Eds.): EvoApplications 2022, LNCS 13224, pp. 667–682, 2022.
https://doi.org/10.1007/978-3-031-02462-7_42

them apply different strategies during the evolution [13] or generate a population of diverse solutions with the hope that some will work as expected in reality [6]. However, it is also possible to avoid simulation and perform evolution directly in hardware. This last approach avoids the reality gap, but faces important challenges.

Varying the morphology of real robots requires a considerable amount of time which could make evolution in hardware unfeasible, specially if each morphology is built from scratch. Current solutions to solving this problem include (1) using specific robotic platforms that can change the length of their limbs and other parts [25], (2) using modular robot platforms [2,19] and (3) combining modular robot systems with 3D printed parts [1,9]. While 3D printing can create robots with almost any shape, it increases the building time and produces non-reusable parts. On the other end, robots that can only change the size of their limbs, although highly reusable, severely restrict the morphological space they can reach. Thus, there is a trade-off on the morphological search space and the reusability of the hardware employed that needs to be balanced when evolving morphologies in hardware [18].

The aforementioned systems make evolution of morphologies in hardware feasible. However, most hardware evolution works ignore the fact that real robot evaluation cannot be parallelized. Evolutionary algorithms are population-based algorithms, and therefore can be parallelized in software easily. However, evolution of morphologies in hardware requires custom and specialized setups that makes having more than one evaluation setup very costly. To the best of our knowledge, all attempts to evolve robotic morphologies in hardware have used only one platform to perform the robot evaluation. Furthermore, the space in these setups is usually reduced and different robots are built with the same basic components, thus it is also not feasible to build and store a population of assembled robots to evaluate them in a future time. This means that each robot needs to be built, evaluated with a controller, and disassembled.

Taking into account the time to build and subsequently disassemble a robot is many times larger than the time required to evaluate it, we propose to use the wall time of the evolution, i.e. the time measured by a wall clock as the evaluations are performed in the real world, as the stop criterion for an evolutionary algorithm rather than the number of evaluations or generations. In this paper, we explore how this strict stop condition affects different strategies that use time more effectively than a traditional strategy for evolving control and morphology of a robot. Specifically, we compare a basic genetic algorithm (GA), used as a baseline, the same GA but testing each morphology with 5 different controllers, and the Edhmor system [7]. The effect of changing the building speed per module on the algorithm results is also analyzed. The experiments have been done in a simulator, but all parameters were chosen to be as close to a physical evaluation as possible.

The next section describes related work that give a higher chance for the controller of a robot to be optimized when optimizing morphology and control with evolutionary algorithms.

## 2 Related Work

Motivated by the fact that morphological changes are destructive [15], several works have suggested to adapt the controller for each morphology in evolutionary robotics. Chocron proposed a nested genetic algorithm for evolving modular manipulators, where the outer algorithm was in charge of the morphological evolution and the inner one obtained a suitable controller for each manipulator [5]. To reduce the number of evaluations needed in the controller adaptation, other authors have investigated a Lamarckian type inheritance, where each robot adapts its controller, based on the controllers inherited from its parents, and passes the optimized controller to its offspring [11]. Similarly, a recent article by Goff et al. proposes keeping an archive of controllers as an inheritance mechanism [8]. Different learning methods for optimizing controllers for different morphologies of modular robots are evaluated in [14], but there is no evolution of morphologies.

All these works try to adapt or learn a controller for each morphology rather than use a joint evolution of morphology and control. Furthermore, all use the number of generations or evaluations to stop the evolution and do not consider the building time of the robots. In addition, most works use an enormous budget of evaluations which is not available when evolving in hardware. In this paper, rather than focusing on optimizing controllers as morphologies change per se, we look at evolution from the perspective of the wall time and how to balance the time spent evaluating new controllers for changing morphologies and the time spent assembling and disassembling morphologies with this strict stop condition.

## 3 Materials and Methods

This section describes the three main aspects that we use for our experiments: The Emerge modules that are used for building the robots, the three different methods used to evolve the morphologies and controllers, and the calculation of the wall clock time.

### 3.1 Emerge Modules

In this paper, we use the Emerge (Easy Modular Embodied Robot Generator) modular robot[1], which is an open source robotic module designed to be easy to build, maintain, and modify [19]. The mating magnetic connectors of the modules allow easy assembly and disassembly of robotic morphologies in seconds either by a human operator or by a robotic manipulator [17]. In addition, magnetic connectors make assembled robots robust against collisions as they can break apart without damaging the modules in case of a collision or if an excessive torque is applied to a connector.

[1] More information about the Emerge robot can be found at https://sites.google.com/view/emergemodular.

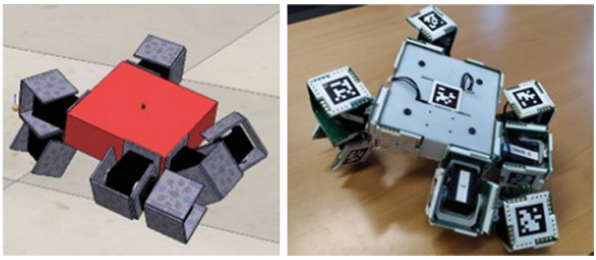

**Fig. 1.** Emerge Modular Robot: the magnetic connections allows a quick assembly of the modules to build a robot, which is useful to evolve morphologies and controllers in reality. An evolved morphology with the base module in the center and several basic modules connected is shown: (right) simulation and (left) reality.

Each module has one servo motor and four connection faces, one of them is connected to the bottom end of the motor and the other three are connected to a bracket, forming a U shape, which is actuated by the shaft of the motor. Connectors in all faces are built with a 3D printed layer and a printed circuit board (PCB) layer. Spring pins are soldered to the PCB layers, which allow module faces to share power and communications. Additionally, a base module with eight connection faces is used as a starting module to build the robots. The base provides a battery and a centralized controller that sends commands to the motors through the motor communication bus while also being able to communicate with an external computer. Both modules are shown in Fig. 1. The basic features of the Emerge modules are described in Table 1. A more detailed description of the Emerge modules can be found in [19].

While the evolution can be performed in a physical setup, we have chosen to carry out the experiments of this paper in simulation to speed up the process and fine tune the algorithms for future hardware runs. Thus, we employ the CoppeliaSim simulator [26] in which the Emerge modules have been already modelled. All parameters of the simulation are set to replicate the physical modules and connections between modules break when facing high torques and forces.

### 3.2 Algorithms

The following three algorithms are tested in this paper with the wall time stopping condition: A genetic algorithm, a genetic algorithm with additional controller evaluations, and the Edhmor algorithm. All of them were implemented using the Java Evolutionary Algorithm Framework (JEAF) [3].

The three algorithms use a tree encoding representation for their individuals that represents the morphology and the controller of a robot. The nodes of the graph represent the modules, and the edges represent the connection between modules. Each node contains the module type, which is fixed in this study (the root node is always a base module and the rest are individual basic Emerge modules) and the parameters of the module controller. An edge contains the face of the parent node where the child is attached and the orientation of the

child (rotation of 0 or 90°). While the modules can be rotated around the center of a connector in multiples of 90°, we only take into account two rotations as the same behaviour can be achieved by adding $\pi$ to the phase offset of the controller.

**Genetic Algorithm (GA).** A genetic algorithm is used to evolve robot morphologies and controllers from the Emerge modules and its results are used as a base to compare the results obtained with the other algorithms. Similar to the methods employed in [23], the genetic algorithm selects parents for crossover using a simple random tournament (with a tournament size of 2) and has two different mutation probability parameters: one for the morphology of an individual and another for each parameter of the controller. The crossover is performed by selecting a random node of both parents (without considering the root node) and swapping their downstream branches. The morphological mutation operator selects one of these operations, each with a 1/3 probability: add a node to any random module with a free face in the robot, change the orientation of a module and the face where it connects to its parent, and delete a random node and its children. All robots go through a mutation of their controller, where each parameter can be mutated with the probability specified. If a mutation occurs, the new value is obtained by adding a Gaussian noise $\mathcal{N}(0, 0.2)$, which is scaled by the range of the parameter, to the old value. If the mutated parameter falls outside a prespecified range, a bounce-back function is used to restore the parameter to its bounds (a circular bounce-back function in case of the phase offset parameter) [22].

A genetic algorithm is expected to take longer to cycle through generations and make a less efficient use of wall clock time when evolving real robot morphologies and controllers as each evaluation encompasses an assembly and disassembly step. All individuals are assembled, tested, and disassembled, even if they have similar or the same morphology. Figure 2 shows the evaluation sequence when evolving robot morphologies and controllers in a genetic algorithm.

**Genetic Algorithm with Additional Controller Evaluations (GA-ACE).** As the basic GA spends most of the evolution time in assembling and disassembling robots, we have also introduced a modified version of the GA that uses the evolution time more effectively. For each robot built, the modified genetic algorithm executes 5 additional evaluations in which only the controller is changed and thus it can also take advantage of already assembled robots to evaluate and optimize their controllers. This kind of evolutionary algorithm is expected to obtain better individuals than a standard genetic algorithm as it can perform more evaluations on the same wall clock time at the expense of slightly reducing the number of different morphologies tested. The 5 additional controller evaluations are obtained by mutating the controller with the same parameters used for the standard GA.

**Edhmor.** We have selected as a third algorithm the Evolutionary Designer of Heterogeneous Modular Robots (Edhmor) [7], which is a custom evolutionary

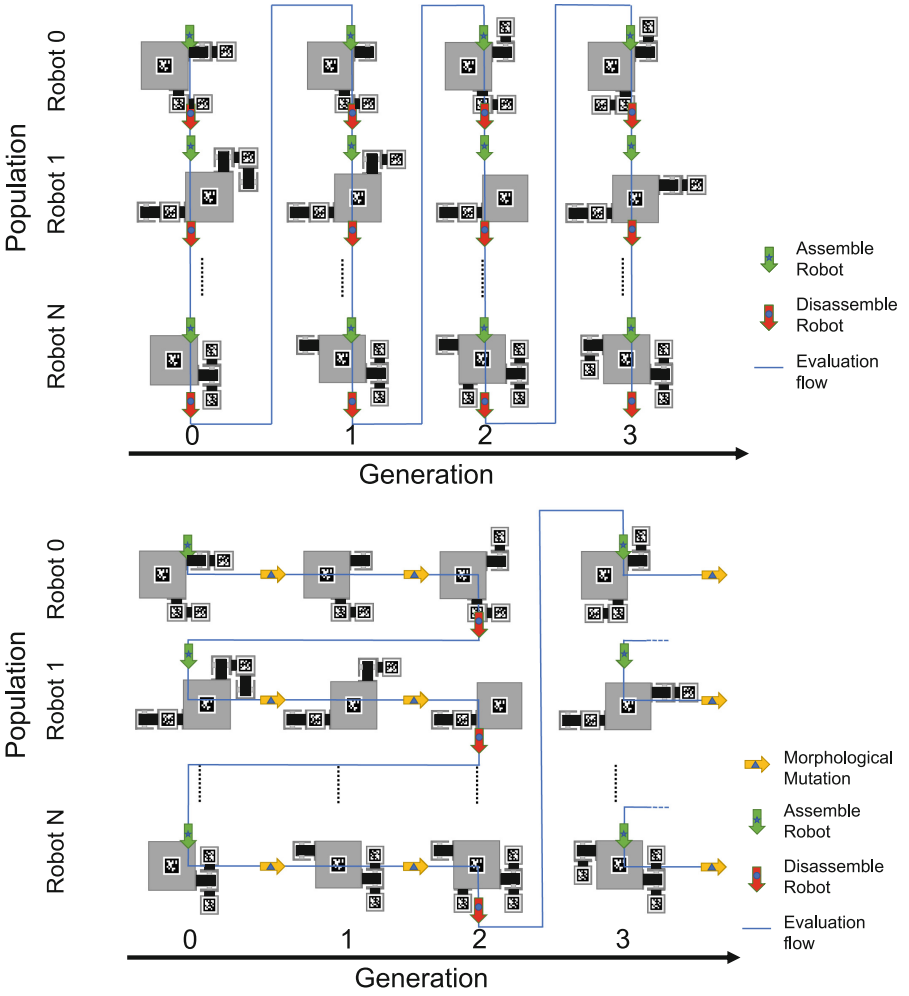


**Fig. 2.** Evaluation sequence of individual robots when evolving robot morphologies and controllers using a genetic algorithm (top) and the Edhmor algorithm (bottom).

strategy to evolve modular robots. Similar to other works [4], the Edhmor system has a simple mechanism to force and protect innovations: It forces morphological innovations by adding modules to the robots in a growing phase, which is followed by other phases where the morphology and controller adapt to the newly introduced modules.

**Table 1.** Simulation experiments parameters

| Parameter | Edhmor | GA | GA-ACE |
|---|---|---|---|
| Max wall clock time (s) | 172800 (48 h) | | |
| Population | 20 | | |
| Repetitions | 40 | | |
| Module assembly time (s/module) | 20,40 | | |
| Morphology mutation probability | N/A | | 0.1 |
| Controller mutation probability | 0.2 | | 0.1 |
| Selection tournament size | N/A | | 2 |
| Settling time (s) | 6 | | |
| Evaluation time (s) | 38 | | |
| Module motor torque (Nm) | 1.8 | | |
| Module actuator range (radians) | $[-\frac{\pi}{2}, \frac{\pi}{2}]$ | | |
| Simulation time step (ms) | 50 | | |
| Physics engine time step (ms) | 5 | | |
| Force sensor torque threshold (Nm) | 1 | | |

We summarize the Edhmor algorithm and specify the tuned parameters that are used in this paper to reduce the number of evaluations and make evolution in hardware feasible. For a more detailed explanation of Edhmor, we refer the reader to [7]. After generating a random population, the following algorithm phases are applied in a loop until the stop criterion is met:

1. *Growing phase* (2 iterations): Adds one child module to a random module of the robot. The orientation and the connection faces between the new module and its parent are generated randomly. The new module is also tested in two additional positions by changing the orientation of the module and where it is connected to its parent. The best of the three robots with the newly introduced module replaces the original robot, even if it is fitness is lower than the fitness of its parent.
2. *Morphological adaptation phase* (2 iterations): A module which is not the root node is selected randomly from the robot. Three new robots are created by mutating the module connection (the parent attaching face and the module orientation). The best of the three robots only replaces the parent if its fitness is better compared to its parent fitness.
3. *Control adaptation phase* (1 iteration): The controller of the robot is mutated six times to generate six new different controllers. The best of the six robots only replaces the parent if the fitness is better compared to that of its parent. The controller mutation operator is the same as in the GA algorithm, but the probability of mutation has been increased, see Table 1.

4. *Pruning phase* (1 iteration): Generates several morphological mutations by iterating over all the modules of the robot (except the root node) and removing them from the robot with their children. If a robot has $M$ modules, this produces $M - 1$ morphological mutations. The best of the pruned robots replaces the parent if its fitness is better compared to that of its parent.
5. *Replacement phase* (1 iteration, immediately run after the pruning phase): Removes the worst 4 individuals and replaces them with 4 individuals produced after applying a symmetry mutation (randomly selecting a limb attached to the root base node and making its reflection through the XZ or YZ planes). Half of the individuals used to produce the mutation are the 2 best robots in the population and the others are chosen randomly. If a symmetry mutation is not possible, a random robot is created instead.

In contrast to previous algorithms, Edhmor keeps the robots assembled between morphological mutations of the same robot across different generations or phases, making only small changes each time (Fig. 2). As Edhmor only needs the fitness of the other robots of the population at the replacement phase, it can apply 6 generations (2 module additions, 2 morphological adaptations, 1 controller adaptation, and 1 prune phase) to the same robot and then change to the next robot of the population (see Fig. 2). Edhmor is thus expected to be able to perform more robot evaluations in the same amount of wall clock time as it does not use as much time assembling and disassembling robots as a genetic algorithm would take.

### 3.3 Wall Time Calculation

We have used a simplified model to calculate the assembly time for each robot based on the time that it takes to assemble one module (MAT), and the robot evaluation time (EVALT). When a population is created and evaluated, the wall time is increased for each robot as shown in Eq. (1).

$$wallTime = \sum_{r=1}^{r=20} (Modules_r * \mathrm{MAT} + \mathrm{EVALT}) \tag{1}$$

Where $Modules$ means the number of modules of a specific robot in the population. In the GA and GA-ACE algorithms, $wallTime$ is increased at each generation by Eq. (2).

$$wallTime = \sum_{r=1}^{r=20} (\mathrm{Modules}_r * \mathrm{MAT} * 2 + \mathrm{EVALT} * (\mathrm{ACE} + 1)) \tag{2}$$

Where ACE is the number of additional controller evaluations (0 for the GA). Notice that this equation takes into account the assembly and disassembly time of the robots.

In Edhmor, the time of each generation depends on the phase that is being performed and is calculated based on Eqs. (3) to (6).

$$wallTime_{grow/morph} = \sum_{r=1}^{r=20} (\mathrm{MAT} * (\mathrm{VAR}_r + 1) + \mathrm{EVALT} * \mathrm{VAR}_r) \quad (3)$$

$$wallTime_{controller} = \sum_{r=1}^{r=20} (\mathrm{EVALT} * \mathrm{VAR}_r) \quad (4)$$

$$wallTime_{prune} = \sum_{r=1}^{r=20} ((Modules_r - \min N) * \mathrm{MAT}) + \mathrm{EVALT} * \mathrm{VAR}_r) \quad (5)$$

$$wallTime_{replace} = \sum_{r=1}^{r=20} (Modules_r * \mathrm{MAT})) \quad (6)$$

Where VAR represents the number of robot variations produced in each phase, and $N$ is the number of modules, thus $\min N$ is the minimum number of modules allowed.

## 4 Experimental Setup

Using the three algorithms, robots are evolved for a locomotion task. In this paper, the controller is kept very simple and each module generates an oscillatory movement where the only parameter that the evolution can adjust is the phase offset. The angle of each joint is controlled by Eq. (7).

$$angle = \frac{\pi}{2} \cdot sin(2 \cdot t + \varphi) \quad (7)$$

Where $\varphi$, the phase offset ($[0, 2\pi)$), is encoded in each individual's chromosome and $t$ is the simulation time. Individual robot solutions are tested by placing them in the center of a simulated flat surface environment and allowing them to move for about 38 s. Simulation is carried out in the CoppeliaSim simulator. The fitness is calculated as the final position of the base module measured in a straight line in the (x, y) plane from the starting position of the robot, as in Eq. (8). The first six seconds of the simulation are not taken into account to discard transient movements of the robot (settling time).

$$Fitness = d((x_{t=38}, y_{t=38}), (x_{t=6}, y_{t=6})) \quad (8)$$

Each method of evolution is run repeatedly 40 times. In the case of the genetic algorithms, a mutation probability of 0.1 is used for both morphological and controller mutations. In all methods used, robots can have a maximum of 16 modules and a minimum of 3 modules. The population is composed of 20 individuals. Robots start always with the flat base module described in Sect. 3.1 as the root node. The initial population is composed of random robots generated

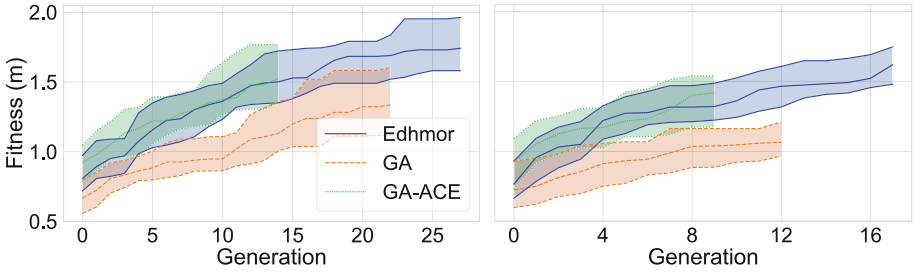


**Fig. 3.** Best individual fitness vs generation in 40 runs of evolving morphology and control with the Edhmor system, a standard genetic algorithm (GA) and a genetic algorithm modified to allow additional controller evaluations (GA-ACE), and allowing (left) 20 s and (right) 40 s of assembly per module. Center line shows the median of each group and the shaded area shows the interquartile range (IQR).

with a maximum of 8 modules attached and a minimum of 3. The assembly time per module is established as a constant and used to register the wall clock time in each algorithm. All algorithms are tested first with a 20 s assembly time per module in the morphology and then with a 40 s assembly time per module (20 s per module is approximately the time that takes to assemble a module manually as reported in [19]). On average, a complete robot has 10 modules and therefore the full assembly time is 200 s in the 20 s per module case when starting to build the robot from scratch. Evolution is stopped after 172800 s (48 h) of simulated wall-clock time have passed.

## 5 Results

Evolution of the best individual fitness over time is often presented as a graph of fitness vs the number of generations that the algorithm performs in the allotted time. Generations represent the cycles of selection, reproduction, and replacement, which in some cases have a direct relation to the wall clock time spent. This style of graph is shown for all evolutionary algorithms used in Fig. 3 and shows the median and interquartile range (IQR) in the case of both using 20 s of assembly time per module and 40s of assembly time per module respectively.

It can be seen that the three evolutionary methods do not achieve the same number of generations. This is a direct consequence of limiting the wall clock time and of the way in which all three evaluate their individuals. It can also be observed that, in the case of using 40 s to assemble each module in a morphology, all methods achieve a fewer number of generations as more time is spent assembling and disassembling robots. In both figures, the GA-ACE method achieves better fitness than the standard GA, even when completing fewer generations.

A more interesting graph can be drawn by changing the X-axis variable from generations to the actual wall clock time spent. This gives us a better picture of how the fitness would change in a real evolutionary run with assembly and

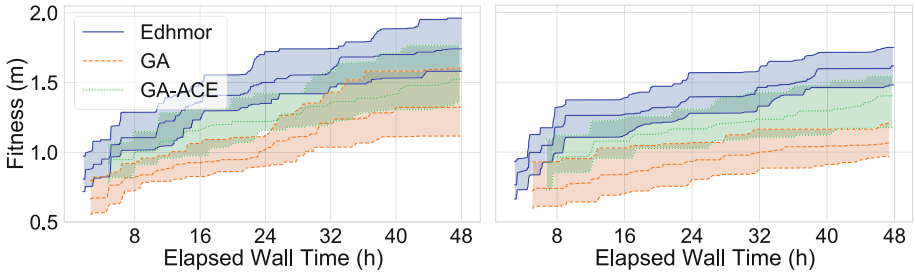


**Fig. 4.** Best individual fitness vs wall clock time (in hours) in 40 runs of evolving morphology and control with the Edhmor system, a standard genetic algorithm (GA) and a genetic algorithm modified to allow additional controller evaluations (GA-ACE), and allowing (left) 20 s and (right) 40 s of assembly per module. Center line shows the median of each group and the shaded area shows the interquartile range (IQR).

disassembly processes. The wall clock time is measured inside each of the evolutionary algorithms as the simulated robots are evaluated (Sect. 3.3), however the exact times can be registered at different moments in different runs due to the different morphologies that appear. As a consequence, and to be able to compare between runs and evolutionary methods, the missing values between runs are interpolated with a linear interpolation. After this process ends, the missing tail values are filled by repeating the closest value.

Figure 4 shows the best fitness in 40 runs of all three evolutionary methods against wall clock time for 20 s of assembly time per module and 40 s of assembly time per module, respectively. Again, the median and interquartile range (IQR) are shown for each group. These figures show that the Edhmor system produces the best individual fitness in the time allotted and at almost all times. The GA-ACE best individuals follow the Edhmor ones and the standard genetic algorithm individuals are the worst performing.

These figures show that going from 20 s to 40 s of assembly time per module increases the separation between the different algorithm groups. A Kruskall-Wallis test showed that there is a statistically significant difference between the fitness of the best individuals of each evolutionary method at the end of all runs, for a 5% significance level (20 s: $p < 0.0001$, 40 s: $p < 0.0001$). A post hoc Dunn test with Bonferroni correction showed that all groups have a statistically significant difference with each other (20 s: all $p < 0.022$, 40 s: all $p < 0.005$).

The better performance achieved by the Edhmor system can be attributed to performing a higher number of fitness evaluations than the other two as was expected. Figure 5 shows the final number of fitness evaluations for each method in 40 runs using 20 s and 40 s of assembly time per module, respectively. Again, as more time is used performing assembly and disassembly processes, runs with 40s of assembly time per module perform fewer fitness evaluations.

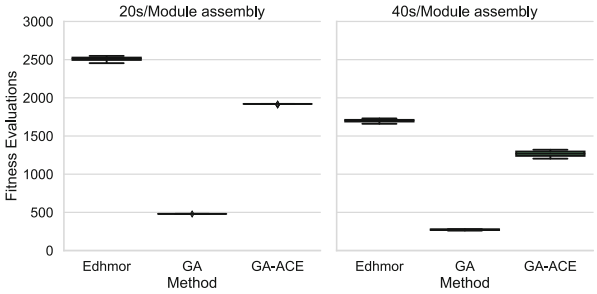


**Fig. 5.** Final number of fitness evaluations performed by each of the evolutionary methods used when using (left) 20 s of assembly time per module and (right) 40 s of assembly time per module.

## 6 Discussion

In this paper, we have investigated the effect of using the wall time as a stopping condition, and taking into account the assembly and disassembly time, when evolving the morphology and controller of robots built using a modular robot platform. Under this strict stopping condition, it was shown that the three algorithms analyzed achieve a different final number of generations and fitness evaluations (Figs. 3 and 5). This is due mainly to how the robot evaluations are organized in each of the three algorithms. Additionally, results show that the time it takes for assembly and disassembly of one module in each morphology directly affects the number of generations and fitness evaluations achieved, which is expected as this time cannot be used to evaluate robots and takes a sizable part of the assigned wall time.

The Edhmor and GA-ACE approaches are shown to produce individuals with better fitness than the standard GA (Fig. 4). In the case of GA-ACE, the increase in fitness is because of the extra evaluations used for optimizing the controller, which coincides with what is observed in other studies that perform controller optimization between morphological changes in evolutionary algorithms [5,8,11]. In fact, this algorithm is the closest evaluated in this paper to an evolutionary algorithm with Lamarckian features as the controller is optimized for a morphology and then transmitted to the robot offspring in the following generation. The main difference is that in our approach the controller mutations are randomly generated and there is no specific algorithm for optimizing the controller apart from the ongoing outer evolutionary algorithm. We could test the influence of using an specific algorithm for optimizing the controller in future work, but we should highlight the minimal budget allowed for controller evaluation (only 6 controller evaluations are tested for each morphology). The emphasis in controller optimization also allows GA-ACE to achieve a higher number of fitness evaluations than the normal GA (Fig. 5), as it does not disassemble and assemble robots as often, and could point to evolutionary algorithms with Lamarckian mechanisms being more efficient when time is limited.

In the case of Edhmor, the increase in fitness can be attributed to two related reasons: First, the Edhmor evaluation flow, keeping the a similar morphology assembled for six generations, allows it to perform a higher number of fitness evaluations in the same time than the other two algorithms (Fig. 2). And second, this evaluation flow keeps making small changes to the morphology of the robot and allows the controllers to adapt to these small changes before removing modules and replacing the worst robots, which intermixes morphology and control changes in a more granular way.

Another advantage of the Edhmor algorithm when working under limited time constrains is that it protects innovations in the morphology of robots similar to the mechanisms presented in [4], as described in Sect. 3.2. This innovation protection can be found also in other evolutionary algorithms as is the case of MAP-Elites [20]. In the MAP-Elites case, innovation protection is achieved by maintaining a diverse population using a grid of desired features. New morphological changes can be stored in one space of the grid and will only be replaced if a better performing robot with similar features is found, which can be the same robot morphology with a better controller. However, depending on the features selected for the grid similar robots can imply big morphological changes thus the performance of MAP-Elites under wall time constrains could vary widely. Nevertheless, MAP-Elites can be used as a first step in simulation to produce a diverse set of robot morphologies and controllers that can later be used as the seed population of a hardware evolution.

Regarding the values selected for assembling one module (20 and 40 s), we believe that they are conservative. 20 s is approximately the time reported in [19] for manual assembly. In [2], a robot of three modules was built by a manipulator in 210 s (70 s per module). In the ARE project, it takes around three minutes to assemble a 3D printed part with three different modules (45 s per component) [10], but excluding the time used for 3D printing. Even in the case where only the length of the limbs is modified without changing the structure of the robot, the change could last up to 90 s [25].

While the results of the paper are not surprising, it is important to notice that few attempts of evolution of robots in hardware have focused on improving the evaluation flow. To the best of our knowledge, none of them have employed evolutionary strategies as a way to reduce the wall time of the evolution [2,8,11,24]. Time is a scarce resource in hardware evolution and algorithms that evaluate similar morphologies consecutively can improve this bottleneck.

Finally, we would like to point out that in this particular case where the time for evolution is severely limited, a technique that can take advantage of the robot evaluation time to improve the controller of a robot would be highly desirable. An example of such a technique is reinforcement learning [12], which works by updating a controller policy based on a reward obtained each time the robot performs an action in the environment. Although of episodic nature, reinforcement learning can be setup to optimize an average reward at each time step of the robot evaluation and thus would be able to optimize the controller of a robot on this smaller time scale. Learning and evolution have been combined

in the past for tackling different tasks and to study the relation between these two processes in reality [21]. A balance would still be needed between the tight time budget that would be allowed for learning, and the time needed to change between morphologies.

## 7 Conclusions and Future Work

In this work, we have studied the effect of using the wall time as a stop condition when evolving the morphology and control of robots built with the Emerge modules. Results showed that the evolutionary algorithms tested were severely constrained by the wall time stop condition. Thus, for evolutionary algorithms to be effective in real hardware tasks, the algorithms must take careful consideration in their use of time. Specifically, they should take into account factors that affect the amount of time that can effectively be used to evaluate robots such as damaged robots, resetting robots to their initial position or building time. In this paper, we have focused on the time necessary for assembly and disassembly robots, which the results showed to directly affect the number of generations and evaluations performed by each algorithm. Therefore, algorithms must be carefully designed to use time efficiently when performing evolutionary robotics experiments in reality, something that is often not considered.

In particular, evolutionary strategies can be more efficient as they can generate several offspring per parent and evaluate them consecutively. In the Edhmor case, small variations in the morphology performed by the morphological mutation phases of the algorithm keep the majority of the robot assembled between changes and as a consequence do not use as much time for assembly and disassembly as the other two algorithms. This allows the Edhmor algorithm to perform the highest number of fitness evaluations overall, and combined with the local search performed by these small variations in morphology and control, also allows it to find the best individuals in the end. From this, it can be concluded that strategies that keep making morphological changes while keeping assembly times at a minimum hold a high potential of obtaining high fitness results when on time constrains.

Future work includes testing the results found in this paper in the real Emerge platform. Furthermore, we would like to explore how the diversity of solutions is affected by the wall time constrain more in detail by creating a pool of diverse solutions in simulation using the MAP-Elites algorithm and using these solutions to seed the Edhmor algorithm. Finally, we would like to test whether using reinforcement learning to optimize controllers when testing different morphologies is effective under a tight evaluation time budget.

## References

1. Auerbach, J., et al.: RoboGen: robot generation through artificial evolution. In: Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems, Artificial Life, pp. 136–137. The MIT Press, New York (2014). https://doi.org/10.7551/978-0-262-32621-6-ch022

2. Brodbeck, L., Hauser, S., Iida, F.: Morphological evolution of physical robots through model-free phenotype development. PLoS ONE **10**(6), 1–17 (2015). https://doi.org/10.1371/journal.pone.0128444
3. Caamaño, P., Tedín, R., Paz-Lopez, A., Becerra, J.A.: JEAF: a Java evolutionary algorithm framework. In: IEEE Congress on Evolutionary Computation, pp. 1–8. IEEE (2010)
4. Cheney, N., Bongard, J., SunSpiral, V., Lipson, H.: Scalable co-optimization of morphology and control in embodied machines. J. Roy. Soc. Interface **15**(143), 20170937 (2018)
5. Chocron, O.: Evolutionary design of modular robotic arms. Robotica **26**(3), 323–330 (2008). https://doi.org/10.1017/S0263574707003931
6. Cully, A., Clune, J., Tarapore, D., Mouret, J.B.: Robots that can adapt like animals. Nature **521**(7553), 503–507 (2015). https://doi.org/10.1038/nature14422
7. Faíña, A., Bellas, F., López-Peña, F., Duro, R.J.: EDHMoR: evolutionary designer of heterogeneous modular robots. Eng. Appl. Artif. Intell. **26**(10), 2408–2423 (2013). https://doi.org/10.1016/j.engappai.2013.09.009
8. Goff, L.K.L., et al.: Morpho-evolution with learning using a controller archive as an inheritance mechanism. arXiv:2104.04269 [cs], September 2021
9. Hale, M.F., et al.: Hardware design for autonomous robot evolution. In: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 2140–2147, December 2020. https://doi.org/10.1109/SSCI47803.2020.9308204
10. Hale, M.F., et al.: The ARE robot fabricator: how to (re)produce robots that can evolve in the real world. In: The 2019 Conference on Artificial Life, pp. 95–102. MIT Press, Cambridge (2019). https://doi.org/10.1162/isal_a_00147.xml
11. Jelisavcic, M., Glette, K., Haasdijk, E., Eiben, A.E.: Lamarckian evolution of simulated modular robots. Front. Robot. AI **6**, 9 (2019). https://doi.org/10.3389/frobt.2019.00009
12. Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: a survey. Int. J. Robot. Res. **32**(11), 1238–1274 (2013). https://doi.org/10.1177/0278364913495721
13. Koos, S., Mouret, J.B., Doncieux, S.: The transferability approach: crossing the reality gap in evolutionary robotics. IEEE Trans. Evol. Comput. **17**(1), 122–145 (2013). https://doi.org/10.1109/TEVC.2012.2185849
14. Lan, G., De Carlo, M., van Diggelen, F., Tomczak, J.M., Roijers, D.M., Eiben, A.E.: Learning directed locomotion in modular robots with evolvable morphologies. Appl. Soft Comput. **111** (2021). https://doi.org/10.1016/j.asoc.2021.107688
15. Lipson, H., Sunspiral, V., Bongard, J., Cheney, N.: On the difficulty of co-optimizing morphology and control in evolved virtual creatures. In: Artificial Life Conference Proceedings 13, pp. 226–233. MIT Press (2016)
16. Marbach, D., Ijspeert, A.J.: Online optimization of modular robot locomotion. In: IEEE International Conference Mechatronics and Automation, vol. 1, pp. 248–253. IEEE (2005)
17. Moreno, R., et al.: Automated reconfiguration of modular robots using robot manipulators. In: 2018 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 884–891, November 2018. https://doi.org/10.1109/SSCI.2018.8628628
18. Moreno, R., Faina, A.: Reusability vs morphological space in physical robot evolution. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, pp. 1389–1391 (2020)
19. Moreno, R., Faiña, A.: EMERGE modular robot: a tool for fast deployment of evolved robots. Front. Robot. AI **8**, 198 (2021). https://doi.org/10.3389/frobt.2021.699814

20. Mouret, J.B., Clune, J.: Illuminating search spaces by mapping elites. arXiv:1504.04909 [cs, q-bio], April 2015
21. Nolfi, S., Bongard, J., Husbands, P., Floreano, D.: Evolutionary robotics. In: Siciliano, B., Khatib, O. (eds.) Springer Handbook of Robotics, pp. 2035–2068. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-32552-1_76
22. Nordmoen, J., Nygaard, T.F., Samuelsen, E., Glette, K.: On restricting real-valued genotypes in evolutionary algorithms. In: Castillo, P.A., Jiménez Laredo, J.L. (eds.) EvoApplications 2021. LNCS, vol. 12694, pp. 3–16. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72699-7_1
23. Nordmoen, J., Veenstra, F., Ellefsen, K.O., Glette, K.: MAP-elites enables powerful stepping stones and diversity for modular robotics. Front. Robot. AI **8**, 56 (2021). https://doi.org/10.3389/frobt.2021.639173
24. Nygaard, T.F., Martin, C.P., Samuelsen, E., Torresen, J., Glette, K.: Real-world evolution adapts robot morphology and control to hardware limitations. In: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO 2018, pp. 125–132. Association for Computing Machinery, New York, July 2018. https://doi.org/10.1145/3205455.3205567
25. Nygaard, T.F., Martin, C.P., Torresen, J., Glette, K., Howard, D.: Real-world embodied AI through a morphologically adaptive quadruped robot. Nat. Mach. Intell. **3**(5), 410–419 (2021). https://doi.org/10.1038/s42256-021-00320-3
26. Rohmer, E., Singh, S.P.N., Freese, M.: V-REP: a versatile and scalable robot simulation framework. In: IROS 2013, pp. 1321–1326. IEEE, Tokyo, November 2013. https://doi.org/10.1109/IROS.2013.6696520
27. Sims, K.: Evolving virtual creatures. In: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH 1994, pp. 15–22. ACM Press, New York (1994). https://doi.org/10.1145/192161.192167