Evolving Modularity in Soft Robots through an Embodied and Self-Organizing Neural Controller

Federico Pigozzi, Eric Medvet
University of Trieste, Trieste, Italy

Abstract

Modularity is a desirable property for embodied agents, as it could foster their suitability to different domains by disassembling them into transferable modules that can be reassembled differently. We focus on a class of embodied agents known as Voxel-based Soft Robots (VSRs). They are aggregations of elastic blocks of soft material; as such, their morphologies are intrinsically modular. Nevertheless, controllers used until now for VSRs act as abstract, disembodied processing units: disassembling such controllers for the purpose of module transferability is a challenging problem. Thus, the full potential of modularity for VSRs still remains untapped. In this work, we propose a novel self-organizing, embodied neural controller for VSRs. We optimize it for a given task and morphology by means of evolutionary computation: while evolving, the controller spreads across the VSR morphology in a way that permits emergence of modularity. We experimentally investigate whether such controller (i) is effective and (ii) allows to tune its degree of modularity, and with what kind of impact. To this end, we consider the task of locomotion on rugged terrains and evolve controllers for two morphologies. Our experiments confirm that our self-organizing, embodied controller is indeed effective. Moreover, by mimicking the structural modularity observed in biological neural networks, different levels of modularity can be achieved. Our findings suggest that the self-organization of modularity could be the basis for an automatic pipeline for assembling, disassembling, and reassembling embodied agents.

1 Introduction

One long-term vision for robotics is building fully autonomous robotic ecosystems (Eiben, 2021). Such robotic communities would cooperate and act to solve tasks that are either too dangerous (e.g., rescue missions, space exploration) or impossible (e.g., digesting pollutants, navigating the vessels of the human body) for humans. In these ecosystems, robots would participate in a pipeline of automatic reconfiguration: as the task and environment change, new robots are "born" and reconfigured from past robots (Buchanan et al., 2020; Hale et al., 2019). In such a pipeline, having a robotic "library" of (pre-optimized) modules

would be desirable for the automatic assembly of new robots. Clearly, modularity plays a crucial role, as it can potentially ease the process of reconfiguring robotic components.

In this paper, we consider Voxel-based Soft Robots (VSRs) (Hiller & Lipson, 2012), aggregations of elastic cubic blocks made of soft material. As such, VSRs are intrinsically modular. VSRs have emerged as a relevant formalism for studying state-of-the-art robotics systems, e.g., soft robotics (Rus & Tolley, 2015). By virtue of their softness, they follow in the steps of natural evolution and resemble natural organisms more closely than their rigid counterparts. They have proved capable of solving challenging tasks, like squeezing through tight spaces (Cheney et al., 2015) and underwater locomotion (Corucci et al., 2018), as well as crossing the sim-to-real gap and designing living organisms (Kriegman, Blackiston, et al., 2020; Kriegman, Nasab, et al., 2020).

Albeit their bodies are intrinsically modular, controllers used until now for VSRs act as abstract, disembodied processing units: disassembling such VSRs for the purpose of reassembling in a different manner, perhaps by combining modules from different VSRs, is a challenging problem. Indeed, modularity is a very desirable property in the road towards fully autonomous robotic ecosystems. Thus, despite the tremendous achievements of VSRs, the full potential for modularity remains unexploited.

To address this dilemma, we here propose a novel self-organizing, embodied Artificial Neural Network (ANN)-based controller for VSRs. Nodes and edges are precisely located throughout the VSR body without a topology fixed a priori. We optimize the controller for a locomotion task (on rugged terrain) and a morphology by means of Evolutionary Computation (EC) (De Jong, 2006): while evolving, the controller spreads and self-organizes across the VSR body in a way that permits emergence of modularity. In fact, self-organization allows to tune the degree of neural complexity across the body. In doing so, we tap into the power of Evolutionary Algorithms (EAs), a family of algorithms that has successfully been applied to the field of robotics (Nolfi & Floreano, 2000; Sims, 1994).

We experimentally inquire whether such controller (i) is effective at solving the task, (ii) is transferable by means of disassembling and reassembling its body and what role modularity plays in the process, and whether (iii) it allows for the automatic discovery of modules.

Our experimental results confirm that our controller is indeed effective. We also find that, when disassembling and reassembling by combining modules from different VSRs, modularity is crucial: more modular controllers turn out to be more transferable than less modular controllers. Finally, our embodied, self-organizing controller allows for the automatic discovery of modules to be transferred. As such, this work positions itself in the road towards an automatic pipeline for the fabrication of new robots in a human-out-of-the-loop manner by assembling modules from other robots.

2 Related work

This work is relevant for research on the topic of modularity in the fields of artificial intelligence and robotics. *Modularity* is an emergent property of a complex system, typically a network. We take inspiration from Yamashita and Tani (2008) and distinguish between structural and functional modularity. In the former, dense connectivity within modules and sparse connectivity between modules emerge. In the latter, a network supports many different functional patterns. For the sake of this study, we are mostly concerned with structural modularity.

From a biological viewpoint, many scientists agree that modularity (the ability to separate functional processes into *modules*) played a key role in the evolvability of natural systems (G. Wagner et al., 2007; G. P. Wagner & Altenberg, 2005). Recently, Gutai and Gorochowski (2021) suggested that modern ANNs are much less modular than biological neural networks, and thus suffer from being *monolithic*. Concurrently, Eiben (2021) put modularity under the spotlight as a key ingredient in the road toward fully autonomous robotic ecosystems. Indeed, Faiña (2021) described the benefits of modularity as simplifying the search space for robotic morphologies and controllers. Additionally, as made clear in Yim et al. (2007), the versatility, robustness, and cheapness of modular robots make them suitable to deployment in a short time.

Under these premises, there has been a growing body of literature devoted to the topic of modular robotics. Starting from the early theoretical formulations (Neumann & Burks, 1966), the last decades saw many physical realizations being proposed (Howison et al., 2021), including soft ones (Sui et al., 2020). Platforms for the automatic design and manufacture of robots from modular components have also been recently explored (Faiña et al., 2015; Moreno et al., 2018). Although interest and progress have been remarkable, there still remain practical and theoretical challenges to be addressed (Liu et al., 2016). In this scenario, our work can be seen as a stepping stone toward reconfigurable robots with non-trivial control.

While the idea of optimizing ANNs by means of EC is definitely not new (de Garis, 1998; Kitano, 1990; K. Stanley et al., 2009; K. O. Stanley & Miikkulainen, 2002), not a lot of studies have investigated the interplay between evolution of ANNs and modularity. Even more, while their achievements have been tremendous, most state-of-the-art neuroevolutionary algorithms fail to produce modular ANNs (Clune et al., 2010). The handful of studies related to evolution and modularity mostly highlight the many benefits of modular ANNs, especially when applied to robotic tasks. In a series of experiments (J. Bongard, 2011; J. C. Bongard et al., 2015; Cappelle et al., 2016), modularity turned out to improve the generalization abilities of robotic agents, and Ellefsen et al. (2015) showed it can also subdue the infamous problem of catastrophic forgetting in ANNs (French, 1999) via reinforcement learning. Other studies highlighted the interlink between modularity and specialization (e.g., multi-tasking), both under a computational (Schrum & Miikkulainen, 2014) and a biological (Espinosa-Soto & Wagner, 2010) perspective. Finally, in a computational biology study,

Clune et al. (2013) demonstrated how modularity can emerge in ANNs as a "spandrel"—i.e., a phenotypic trait that is a byproduct of evolution of some other trait (Gould & Lewontin, 1979)—to minimize connection costs between neurons.

Despite being ground-breaking, these works suffer from one (or both) of the following pitfalls. First, only some of them (J. C. Bongard et al., 2015; Cappelle et al., 2016) are conducted under the embodied cognition paradigm, which postulates that intelligence emerges from the complex interactions between the brain, the body, and the environment (Brooks, 1990; Pfeifer & Bongard, 2006). The others consider ANNs that function as abstract processing units, and are not pervasive with respect to the body they control. Indeed, Mitchell (2021) listed a lack of embodiment as one of the four "fallacies" of artificial intelligence, and the dualist bias "body vs. mind" is well-known to be rooted in our culture (Bloom, 2004). Second, even when under an embodiment perspective, none of them addresses how to self-organize modularity inside a robotic agent body and exploit it for the sake of reconfigurability.

In this work, we build on our previous research in the field of evolutionary robotics. In (Medvet, Bartoli, De Lorenzo, & Fidel, 2020), we proposed a representation for a distributed and embodied controller and a reconfigurability procedure for VSRs that, together, proved effective. For the sake of this study, we term that procedure disassembly-reassembly and describe it thoroughly in Section 5.2.1. Later, in (Medvet et al., 2021), we showed how that representation could be made more compact without loss of effectiveness or diversity. We extend our previous studies by introducing a self-organizing controller that is capable of supporting differing degrees of complexity across the body; as a result, it is better suited to further exploit the intrinsic modularity of VSRs.

3 Background: Voxel-based Soft Robots

Voxel-based Soft Robots (VSRs) are a kind of modular robots composed as aggregations of elastic cubic blocks (*voxels*), made of soft material. Each voxel acts by contracting or expanding its volume and it is the overall symphony of volume changes that allows for the emergence of behavior at the robot level. VSRs were first formalized in Hiller and Lipson (2012), together with a fabrication method. In this work, we consider a 2-D variant of simulated (in discrete time and continuous space) VSRs (Medvet, Bartoli, De Lorenzo, & Seriani, 2020a). While disregarding one dimension certainly makes these simulated VSRs less realistic, it also eases the optimization of VSR design, thanks to the smaller search space. We remark, however, that the representation and the algorithms adopted in this paper are easily portable to the 3-D setting.

In the following, we outline the characteristics of VSRs relevant to this study, and refer the reader to (Medvet, Bartoli, De Lorenzo, & Seriani, 2020a, 2020b) for more details. A VSR is completely defined by its *morphology* (i.e., the body) and its *controller* (i.e., the brain). The former is in turn built with a *shape*, dictating how many voxels the robot is constructed with and how they

are arranged in a 2-D grid, and a *sensory apparatus*, telling what are the sensors and how they are placed over the robot body. Sensors can provide information about the external environment and the robot itself, a capability that has been shown to be effective for locomotion (Talamini et al., 2019). The controller is in charge of determining how the area of each voxel varies over time, based on the sensor readings.

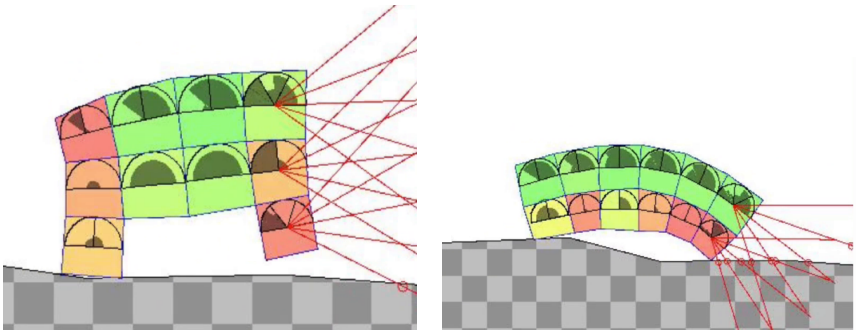
3.1 VSR morphology

The morphology of a VSR describes how the voxels, i.e., deformable squares of side length $l = 3\,\text{m}$, are arranged in a grid topology of size $w \times h$. Each voxel is modeled as the assembly of spring-damper systems, masses, and distance constraints (Medvet, Bartoli, De Lorenzo, & Seriani, 2020a) and is rigidly connected to its four adjacent voxels (if present). We set the same parameters for the components of each voxel as the default ones; as a result, all the voxels share the same mechanical properties.

Over time, being elastic soft blocks, the voxels change their area according to (a) external forces acting on the voxel (e.g., other voxels and bodies like the ground) and (b) a control signal dictated by the controller. The latter produces a contraction/expansion force that is modeled in the simulation as an instantaneous change in the resting length of the spring-damper systems of the voxel. The length change is linearly dependent on an *actuation value* residing in $[-1, 1]$, -1 being the greatest possible expansion and 1 being the greatest possible contraction. The controller sets the actuation value for each voxel, at every time step of the simulation.

A VSR can be equipped with sensors, and each voxel can have one or more sensors. A sensor outputs, for every time step, a *sensor reading* $\boldsymbol{s} \in \mathbb{R}^m$, where m is the dimensionality of the sensor type. In this study, we equip VSRs with four different types of sensor. Area sensors sense the ratio between the current area of the voxel and its resting area ($m = 1$). Touch sensors sense whether the voxel is currently touching another body different from the enclosing VSR (e.g., the ground) or not, and output a value of 1 or 0, respectively ($m = 1$). Velocity sensors sense the speed of the center of mass of the voxel along the x- and y-directions ($m = 2$). Lidar sensors sense the distance to the closest objects along a predefined set of directions. Precisely, for each direction, a lidar sensor measures the distance between the voxel center of mass and the closest object in that direction, clipping it to d. If no object is present at all, the sensor reading is set to d. We used $d = 10\,\text{m}$ and the following directions with respect to the positive x-axis: $-\frac{1}{4}\pi$, $-\frac{1}{8}\pi$, 0, $\frac{1}{8}\pi$, $\frac{1}{4}\pi$ (so $m = 5$). Sensor readings undergo a soft normalization, with tanh function and rescaling, to ensure the output is in $[0, 1]^m$. After normalization, every sensor reading $\boldsymbol{s}$ is perturbed into $\boldsymbol{s}' = \boldsymbol{s} + \boldsymbol{\nu}$, with $\boldsymbol{\nu} = \{\nu_i\}_i \in \mathbb{R}^m$ and $\nu_i \sim \mathcal{N}(0, \sigma^2_{\text{noise}})$ being additive Gaussian noise of mean 0 and variance σ^2_{noise}. We set $\sigma_{\text{noise}} = 0.01$. The purpose of this transformation is to simulate real-world sensor noise and, indirectly, to favor generalization ability in the evolved controllers.

Figure 1 shows two example VSRs simulated using our software.



(a) Biped (b) Worm

Figure 1: A biped and a worm example morphologies, taken at a snapshot of the simulation. Each square is a voxel. The color represents the ratio between its current area and its rest area: red stands for contraction, green for expansion, yellow for no change. The semi-circular sectors drawn at the center of each voxel encode the sensor readings, and are partitioned into subsectors according to the number of sensors; subsectors are further partitioned according to the sensor dimensionality m. The red lines depict the rays of the lidar system.

3.2 VSR controller

The controller is the main focus of this study. Let n be the number of voxels of the VSR and let $\boldsymbol{r}^{(k)} = [s_1 \ s_2 \ \dots]$ be the concatenation of sensor readings for all the VSR sensors at time step k, i.e., at time $t = k\Delta t$, where Δt is the interval between two simulation time steps. The controller is a dynamical system with input $\boldsymbol{r}^{(k)}$ and output $\boldsymbol{a}^{(k)} \in [-1, 1]^n$.

Previous works dealing with VSRs employed different approaches in instantiating this general definition, with different degrees of complexity. In some works, as, e.g., (Cheney et al., 2013), the controller output $\boldsymbol{a}^{(k)}$ depends only on k, i.e., it does not exploit the information coming from the sensors. In other cases, as, e.g., (Talamini et al., 2019), the controller does not have a state, i.e., $\boldsymbol{a}^{(k)}$ depends only on the current $\boldsymbol{r}^{(k)}$. In the latter scenario, the controller can be modeled as a function $\boldsymbol{a}^{(k)} = f\left(\boldsymbol{r}^{(k)}\right)$, or simply $\boldsymbol{a} = f(\boldsymbol{r})$. In this condition, and if the function $f : \mathbb{R}^p \to \mathbb{R}^n$ can be parametrized with a numerical vector $\boldsymbol{\theta} \in \mathbb{R}^q$, then the problem of finding a good controller given a morphology and a task can be cast as a numerical optimization problem.

3.3 Limitations of most controllers

Several kinds of controllers have proven effective for robotics tasks, including phase controllers (Joachimczak et al., 2016), central pattern generators (Kamimura et al., 2004), and ANNs (Ferigo, Iacca, Medvet, & Pigozzi, 2021; Talamini et al., 2019). Nevertheless, the majority of them are disembodied and do not self-organize, suffering from limitations that we detail here.

First and foremost, they do not allow for the full exploitation of modularity and reusability. Consider the case of disassembling a VSR for the sake of transferring its components to other robots and reusing them. A disembodied controller, being tightly bound to the morphology, would require to be re-designed from scratch. Indeed, the difficulties controllers face when adapting to unseen morphologies have been documented by Lipson et al. (2016). In fact, a learned controller is coupled with the overall VSR morphology and how it interacts with the environment, consistently with the embodied cognition paradigm (Pfeifer & Bongard, 2006). Once transferred to a new morphology, controllers have to be learned all over again, resulting in the notorious "catastrophic forgetting" problem (French, 1999), i.e., controllers learned for one problem become ineffective when transferred to a new one.

Second, they usually do not self-organize their structure, shrinking the space of possible functions that can be explored. Self-organization is beneficial as it can discover emergent properties that cannot arise by virtue of a single intelligence only. In nature, self-organization provides mechanisms that evolution can exploit (Johnson & Lam, 2010) and instances of self-organizing intelligence are countless in biology.

Our controller addresses all of these shortcomings by virtue of its embodiment and self-organization properties, as we explain in the following.

4 Embodied, self-organizing neural controller

4.1 Design goals

Based on the considerations drawn in Section 3.3, we want our controller to be:

(a) Embodied: it has to be located in the robot body.

(b) Self-organizing: it must permit different degrees of complexity in different parts of the body. The actual distribution of complexity that fits a given task and morphology must emerge from optimization.

(c) Optimizable for a given task and morphology.

4.2 Definition

Based on the vast existing literature on ANNs, which are well-known universal function approximators (Goodfellow et al., 2016; Schäfer & Zimmermann, 2006), and considering previous applications of ANNs for controlling VSRs (Ferigo, Iacca, & Medvet, 2021; Medvet et al., 2021; Talamini et al., 2019), we decided to rely on ANNs for building our embodied, self-organizing controller.

We represent the controller as a directed graph $G = (V, E)$ encoding an ANN where V are the nodes and E are the edges. The nodes of the graph are the neurons, and the edges of the graph are the synapses between them describing how computation flows over the ANN.

Each node is a tuple $v \in \mathbb{N} \times \{0, \ldots, w-1\} \times \{0, \ldots, h-1\} \times \mathcal{T}$, where the elements of the tuple constitute the node attributes that we denote using the following dot notation, for the sake of clarity. $v.\text{index} \in \mathbb{N}$ is the unique identifier for the node in the graph and its sole purpose is to formally permit the existence in V of nodes with the same values for all the other attributes. $v.\text{x} \in \{0, \ldots, w-1\}$ and $v.\text{y} \in \{0, \ldots, h-1\}$ are the coordinates of the voxel the node is placed into—recall that a VSR morphology is a grid of size $w \times h$. $v.\text{type} \in \mathcal{T} = \{\text{SENSOR}, \text{HIDDEN}, \text{ACTUATOR}\}$ is the neuron type. Each node of type SENSOR is statically associated with one element of one sensor placed in the same voxel of the node. Similarly, each node of type ACTUATOR is statically associated with the actuator of voxel the node is in.

Each edge is a tuple $e \in V \times V \times \mathbb{R} \times \mathbb{R}$. $e.\text{source} \in V$ and $e.\text{target} \in V$ are the source and target nodes, respectively. $e.\text{weight} \in \mathbb{R}$ is the edge weight. $e.\text{bias} \in \mathbb{R}$ is the edge bias.

Since each one of the nodes of V is located in a precise voxel of the morphology, the controller is embodied, which meets our first design goal. As a further consequence, nodes and edges can be located more or less densely across the morphology, which meets our second design goal.

4.3 Computation

During the simulation, at each time step the controller reads the readings from the sensors and inputs them to SENSOR neurons, propagates the values across HIDDEN neurons, and outputs as actuation values the values that reached the ACTUATOR neurons.

In detail, the controller works as follows. Let $\text{in}(v) \subseteq E$ be the set of incoming edges of a node v. Let $h^{(k)}(v) \in \mathbb{R}$ be the *activation value* of node v at step k. Collectively, the activation values $\{h^{(k)}(v)\}_{v \in V}$ of all the neurons $v \in V$ constitute the *state* of the controller at step k. Initially, at $k = 0$, we set each activation value to 0. Then, we update the state as follows. If the node is a SENSOR neuron, then we set $h^{(k)}(v)$ to the tanh of the element of the current reading of the sensor associated with the node. Otherwise, we set the activation value to:

$$h^{(k)}(v) = \tanh\left(\sum_{e \in \text{in}(v)} h^{(k-1)}(e.\text{source}) \cdot e.\text{weight} + e.\text{bias}\right) \tag{1}$$

The concatenation of the activation values of the nodes of type ACTUATOR forms the actuation values $\boldsymbol{a}^{(k)}$ at step k.

In other words, at time step k every node sends the activation value to all its outgoing neighbors. At the next time step $k+1$, the neighbors, in turn, apply the activation function to it and save the result as the activation value for the next time step. Every edge propagates a message only once per time step, at every time step. Having no real incoming neighbors, a SENSOR node treats the sensor reading as if it was the message propagating from an incoming neighbor.

We remark that this controller is a dynamical system, since there is a delay of one time step in the propagation of information along each edge. Moreover, we remark that the controller representation does not forbid cycles in the graph.

4.4 Optimization

We want to optimize a controller, i.e., a graph as defined in Section 4.2, for a given task and a given robot morphology.

Graphs are, in general, difficult to optimize: being the search space non-numeric, we cannot employ standard numerical optimization algorithms. We resort to EC for optimization. Indeed, EAs have already proven capable of dealing with graph-like structures (Medvet & Bartoli, 2020; Miller & Harding, 2008; K. O. Stanley & Miikkulainen, 2002), provided that a fitness function and an appropriate representation are given. Additionally, EAs have also been argued to be a competitive alternative when optimizing ANNs for continuous control tasks (Such et al., 2017). EC is thus suitable for achieving the third design goal.

We used the EA in Algorithm 1. It evolves a fixed-size population of n_{pop} individuals, i.e., graphs, initially set randomly, for a fixed number of n_{gen} generations. At each generation, the current population is used to generate an offspring of n_{pop} new individuals that are then merged into the parent population. Each new individual G' is generated by applying mutation to a parent G selected with tournament selection with size n_{tour}. After parents and offspring have merged, the population for the next generation is selected by picking the n_{pop} best individuals, i.e., using truncation selection.

```
1  function evolve():
2      P ← initialize(n_pop)
3      foreach i ∈ {1, ..., n_gen} do
4          P' ← ∅
5          foreach j ∈ {1, ..., n_pop} do
6              G ← selectTournament(P, n_tour)
7              G' ← mutate(G)
8              P' ← P' ∪ {G}
9          end
10     end
11     P ← P ∪ P'
12     P ← truncate(sort(P), n_pop)
13 end
```

Algorithm 1: The EA used in our experiments.

Since the controller consists in a graph, we used genetic operators suitable for this kind of representation. In particular, we used five mutation operators: edge mutation, edge addition, edge removal, node addition, and node removal. In line 7 of Algorithm 1, we beget an offspring G' by first copying the parent

G and then applying one of the five mutation operators according to their relative probabilities p_{mut}, p_{edgeAdd}, $p_{\text{edgeRemove}}$, p_{nodeAdd}, and $p_{\text{nodeRemove}}$. In the following, we detail the five mutation operators:

- *Edge mutation.* We randomly pick an edge $e \in E$ with uniform probability and mutate its weight and bias with additive Gaussian noise, i.e., $e.\text{weight} \leftarrow e.\text{weight} + \alpha$ and $e.\text{bias} \leftarrow e.\text{bias} + \beta$ with $\alpha, \beta \sim \mathcal{N}(0, \sigma^2_{\text{mut}})$.

- *Edge addition.* We randomly pick two nodes $u, v \in V$ with uniform probability, such that $u.\text{type} \neq \text{ACTUATOR}$ and $v.\text{type} \neq \text{SENSOR}$. Then, we add an new edge e from u to v, i.e., with $e.\text{source} = u$ and $e.\text{target} = v$, and we set $e.\text{weight}, e.\text{bias} \sim \mathcal{U}(-1, 1)$.

- *Edge removal.* We randomly pick an edge $e \in E$ with uniform probability and delete it from the graph. If, after this operation, there are hidden nodes having no incoming and outgoing edges left, we remove them from V to decrease redundancy of the representation (Rothlauf, 2006; Rothlauf & Goldberg, 2003).

- *Node addition.* We randomly pick two nodes $u, v \in V$ with uniform probability, such that $u.\text{type} \neq \text{ACTUATOR}$ and $v.\text{type} \neq \text{SENSOR}$. We then create a new hidden node w and set $w.\text{x}$ and $w.\text{y}$ to be the coordinates of a voxel picked at random. Finally, we add an edge e from u to w and an edge e' from w to v and we set $e.\text{weight}, e.\text{bias}, e'.\text{weight}, e'.\text{bias} \sim \mathcal{U}(-1, 1)$.

- *Node removal.* We randomly pick a hidden node $v \in V$ with uniform probability and delete it from the graph, together with all its incoming and outgoing edges. If, after this operation, there are hidden nodes having no incoming and outgoing edges left, we remove them from V.

We initialize each individual of the initial population as follows. We add to v a sensor node for each sensor reading element in the given morphology, associate v the node with it, and set $v.\text{x}$ and $v.\text{y}$ to the coordinates of the voxel hosting the sensor. We add to V an actuator node v for each voxel in the given morphology, associate v the node with the actuator, and set $v.\text{x}$ and $v.\text{y}$ to the coordinates of the voxel. For each actuator node v and each sensor node u in the same voxel, we add to E an edge e from u to v and we set $e.\text{weight}, e.\text{bias} \sim \mathcal{U}(-1, 1)$.

As a result of this initialization procedure, there are no edges encroaching on different voxels for the VSRs of the very first generation, and the controllers can be regarded as being minimal. We adopt this initialization strategy in line with the complexification principle of starting minimally and incrementally growing topologies of ANNs (K. O. Stanley & Miikkulainen, 2002).

We remark that the mutation operators that affect the topology of the graph, i.e., edge and node addition (or removal), by changing the preference for edges inside or outside modules, can affect the degree of modularity of the controller. This consideration will reveal itself useful in Section 5.2.

We used the same parameter values for all experiments, with values determined after preliminary experiments, with the exception of n_{gen} that is detailed separately for every experiment. We set $n_{\text{pop}} = 96$, $n_{\text{tour}} = 5$, $\sigma_{\text{mut}} = 0.7$, $p_{\text{mut}} = 0.5$, $p_{\text{edgeAdd}} = 0.1$, $p_{\text{edgeRemove}} = 0.1$, $p_{\text{nodeAdd}} = 0.15$, and $p_{\text{nodeRemove}} = 0.15$.

4.5 Advantages and limitations of the embodied, self-organizing controller

The controller presented in this section addresses the design goals presented in Section 4.1 and has several advantages.

First, as required by the first design goal, it is *embodied*, as the nodes are precisely located throughout the VSR body. By virtue of this property, the VSR brain can be partitioned into subgraphs (by selecting subsets of nodes and the edges incident to them), and each of the subgraphs can then be traced back to the portions of body it belongs to. At the same time, should the VSR being dissected into physical modules (e.g., for the sake of transferring it to assemble a new VSR), each physical module would then be able to refer to the subgraph of the controller it was the embodiment for. As a result, each subgraph (once transferred with its physical counterpart) still finds itself in a body it is acquainted with, and is well-positioned to exploit the representations learned during its "previous life". Other studies have indeed highlighted how modular controllers are less prone to catastrophic forgetting (Ellefsen et al., 2015). Since functional modules, i.e., units sharing the same function or purpose, can arise in embodied agents, the representations learned locally in a given functional module could well be transferred to a new module performing the same, or a similar, function.

Since both the topology and the parameters can freely evolve in the body, our controller is *self-organizing* and thus satisfies the second design goal. This fact bears an important consequence: emergent properties, that would not arise without self-organization, can be observed. This is a consequence of the expressiveness of our representation: the search is performed in the space of every possible topology (and parametrization) encompassing a given morphology. We remark that there is no upper (or lower) bound on the length (measured, for example, in terms of voxels to be crossed) of the edges. Potentially, interesting long-range synapses can be established between the neurons of the controller (e.g., between the two legs of the biped shape presented in Figure 1). Remarkably, self-organization can also be biased purposely in order to explore some portions of the search space more densely, with the goal of easing the emergence of some properties we deem interesting a priori. Moreover, self-organization could be used as a way of testing how the "brain" of a robot adapts with respect to the sensors and the robot shape, and make fascinating parallels with neuroscience. As intriguing as it may be, we did not explore this last topic further, and leave it for future work.

Third, messages propagate over the graph one hop at a time, and these delays could be exploited by the optimization algorithm to introduce timed

dependencies that would not be possible otherwise. As a proof of concept, Cheney et al. (2014) demonstrated the benefits of having messages propagating "physically" across a VSR body and the claim is biologically grounded (Segev & Schneidman, 1999). Moreover, we remark our model is *stateful*, and could thus show interesting temporal dynamic behavior, as it couples two dynamical systems, the mechanical one and the controller one. Contrarily, a stateless controller (without cycles and delays) couples its own static system with only the mechanical dynamical system.

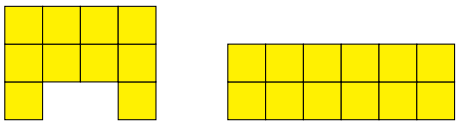
Fourth, *distributing* the controller delivers advantages on its own. Distributed controllers do not suffer from having a single point-of-failure, and would thus prove resilient to both exogeneous (e.g., environmental changes) and endogeneous (e.g., malfunctions) threats once deployed *in vivo*. As a biological counterpart, animals that excel in regenerating their amputated limbs, like sea stars (Lawrence, 2020), usually have a distributed nervous system[1]; some species can even beget stand-alone new individuals from fragments of their bodies, like flatworms of the genus *Planaria* do (Gentile et al., 2011). Although there is evidence that these properties are due to pluripotent stem cells (Baguna et al., 1989), having a less centralized nervous system intuitively eases regeneration.

We remark that the way we model the embodied, self-organizing controller in this study also implies a few limitations.

First, the location of nodes within voxel is actually relevant only for nodes of types ACTUATOR and SENSOR. Since there are no differences in the way the activation value is computed between pair of nodes located in the same or different voxel, a controller is in practice functionally invariant with respect to the location of HIDDEN nodes. In order to verify the practical impact of this modeling choice, we conducted an experimental campaign where we explicitly took into account the location of HIDDEN nodes too as follows. We modified the edge addition genetic operator in such a way that long connections had a lower probability of being added: as a consequence, the location of all nodes—including HIDDEN nodes—matter when evolving the controller topology. We do not present the full results for brevity, but we found that the controllers evolved in this way were in general worse than those evolved without considering nodes location. As an alternative, we could take into account node location also by introducing longer delay in information spreading along long connections: we leave this possibility for future work.

Second, we do not model the cost of complex and large controllers. As a consequence, there are no factors driving the evolution towards parsimony: we expect the complexity of the network to increase virtually unbounded during the evolution. In fact, this is what we observed in our experiments. While we acknowledge that the physical realization of arbitrarily complex neural controllers might be unfeasible, the size of the controllers we evolved in this study is in practice small, in the order of few hundreds of nodes and edges.

[1]Notable exceptions being salamanders (Joven et al., 2019) and axolotls (Roy & Gatien, 2008).



(a) Biped (b) Worm

Figure 2: The two morphologies used in our experiments.

5 Experimental evaluation

We performed several experiments aimed at answering experimentally the following research questions:

RQ1 Is the proposed controller effective?

RQ2 Does it enable module transferability? Can transferability be favored or disadvantaged?

RQ3 Can we automatically discover modules?

For answering these questions, we performed several experiments with two different VSR morphologies, depicted in Figure 2. We experimented with a 4×3 (size of the voxel grid constituting the shape) rectangle with a 2×1 rectangle of missing voxels at the bottom-center, that we call *biped*, and with a 6×2 rectangle, that we call *worm*. We use similar sensor configurations for the two shapes: area sensors for every voxel, touch sensor for the voxels in the bottom row of the shape, velocity sensors for the voxels in the top row of the shape, and lidar sensors for the voxels in the rightmost column of the shape. This sensor configuration results in an overall number of $31 = 10 \cdot 1 + 2 \cdot 1 + 4 \cdot 2 + 3 \cdot 5$ sensor nodes and 10 actuator nodes in the controllers for the biped and $40 = 12 \cdot 1 + 6 \cdot 1 + 6 \cdot 2 + 2 \cdot 5$ sensor nodes and 12 actuator nodes in the controllers for the worm.

For all the experiments, we considered the task of locomotion. The goal of the VSR is to travel as far as possible on a terrain along the positive x direction in a fixed amount of simulated time. The fitness of the individual is the average velocity, measured as:

$$\bar{v}_x = \frac{x_c(t_{\text{final}}) - x_c(t_{\text{transient}})}{t_{\text{final}} - t_{\text{transient}}}, \tag{2}$$

where $x_c(t)$ is the x-position of the center of mass of the VSR at time t. We set $t_{\text{final}} = 30\,\text{s}$ and $t_{\text{transient}} = 5\,\text{s}$ to discard the initial transitory phase and avoid deceptive and inconclusive behaviors (Whitley, 1991). Locomotion is a classic task in evolutionary robotics and usually consists in making the robot run along a flat surface. We here used instead an uneven (hilly) terrain with bumps. The bumps have an average height of 1 m (3 m being the side length of a voxel) and an average distance of 10 m. Moreover, we used a different hilly randomly-generated terrain at every fitness evaluation, so as to prevent the robots from "overfitting" to a single terrain profile, making adaptation more challenging.

We implemented the software for the experimental evaluation in Java, building on two frameworks: JGEA[2] for the evolutionary optimization and 2D-VSR-Sim[3] (Medvet, Bartoli, De Lorenzo, & Seriani, 2020a) for the simulation of VSRs. In the simulations, we set the time step to $\Delta t = \frac{1}{60}$ s and all other parameters to default values. The code for the experiments is publicly available at https://github.com/pigozzif/SelfOrganizingVSRController.

For each experiment, unless otherwise specified, we performed 10 evolutionary runs by varying the random seed for the EA. We remark the each simulation is instead deterministic, given a terrain and a VSR. After verifying the adequate hypotheses, we carried out all statistical tests with the Mann Whitney U rank test for independent samples (Mann & Whitney, 1947) using, unless otherwise specified, 0.05 as confidence level.

5.1 RQ1: effectiveness of evolved controllers

We say that a controller is effective if it meets two criteria: (a) it can successfully solve the task at hand and (b) it is clearly more complex than the minimal controller. The latter point is crucial, since a controller that has not evolved to be more complex than the minimal controller could not be considered "self-organizing", invalidating all of our conjectures. We thus introduce a simple, yet effective measure of controller complexity that measure its *size*:

$$|G| = |V| + |E| \tag{3}$$

that is the sum of the number of nodes and edges of the graph. Concerning the success in solving the task, we measure it using the robot average velocity $\bar{v}_x$, as defined in Equation (2).

We ran an experimental campaign with 10 runs lasting $n_{\text{gen}} = 3000$ generations each, and for the two shapes introduced previously.

Figure 3 shows, for the two shapes, the best individual velocity $\bar{v}_x$ and controller size $|G|$ during the evolution. For both indexes, the figure shows the median $\pm$ standard deviation across the 10 runs.

Table 1 reports the results in terms of $|G|$, and its components $|V|$ and $|E|$, for the best individual at the last generation, separately for the two shapes, and compares them with the values at the start of evolution. Recall that, for a given shape, $|G|$, $|V|$, and $|E|$, are the same for every individual in the first generation (given a morphology). For the indexes at the last generation, the table shows the median and standard deviation across the 10 runs.

From the table and the figure, we observe that the evolved VSRs turn out to be effective according to our criteria. First, evolution is capable of finding good solutions to the locomotion task. In fact, the best individuals run on average at $\bar{v}_x = 5\,\text{m/s}$ at the end of evolution, against approximately $0.5\,\text{m/s}$ at the beginning, signifying a considerable gain in locomotion skills. We visually inspected the behaviors and found them to be highly effective for locomotion

[2]https://github.com/ericmedvet/jgea
[3]https://github.com/ericmedvet/2dhmsr

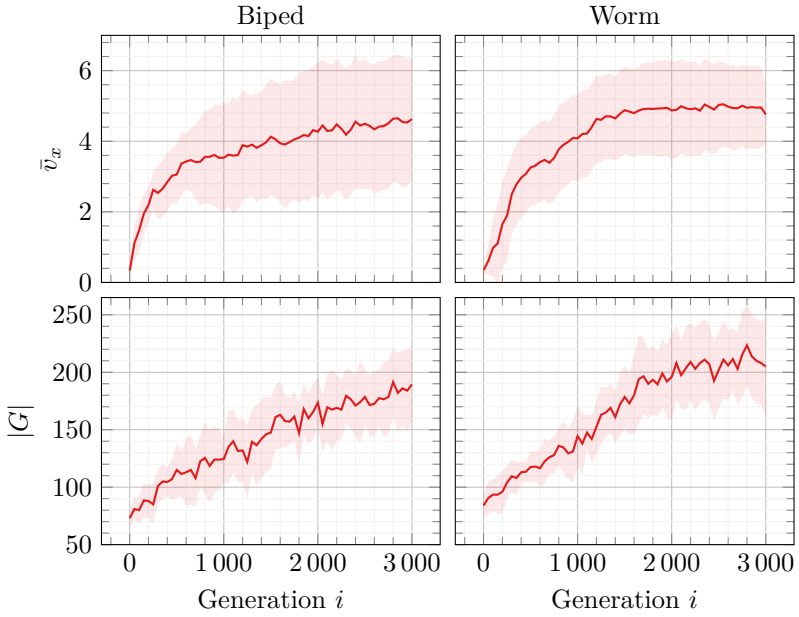
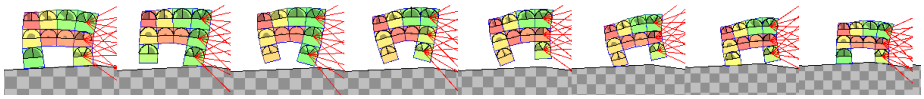


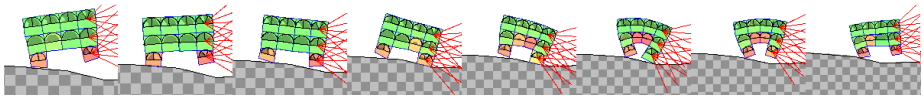
Figure 3: Median ± standard deviation (solid line and shaded area) across the 10 runs of the robot velocity $\bar{v}_x$ (top) and controller size $|G|$ (bottom) for the best individual, for biped (left) and worm (right).

	Initial			At last generation		
Shape	$\|V\|$	$\|E\|$	$\|G\|$	$\|V\|$	$\|E\|$	$\|G\|$
biped	45	35	80	88.5±17.3	96.5±26.8	179.5±42.2
worm	52	40	92	99.0±16.6	115.0±24.8	216.0±39.4

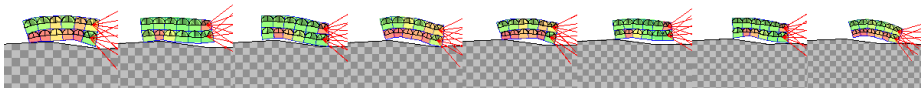
Table 1: Median and standard deviation of number of nodes $|V|$, number of edges $|E|$, and controller size $|G|$ for the best individual, at initialization and at the last generation.



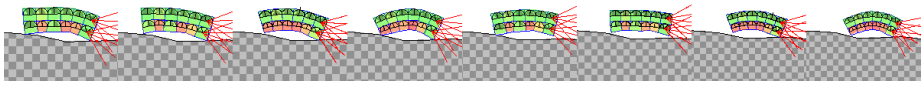
(a) Biped (video available at https://youtu.be/qkVa9hiOveI)



(b) Biped (video available at https://youtu.be/oYeaZwtR8OI)



(c) Worm (video available at https://youtu.be/k1f2vQNyrEU)



(d) Worm (video available at https://youtu.be/HaZHnaRIPqA)

Figure 4: Time-lapse showing locomotion for two bipeds and two worms chosen randomly among the 10 best individuals resulting for each of the two morphologies.

on a hilly terrain. As a proof of concept, Figures 4a to 4d are time-lapse images for the movement of two bipeds and two worms chosen randomly among the 10 best individuals resulting for each of the two shapes. As can be seen from the pictures, bipeds hop on their legs, and worms inch forward as a caterpillar would do. To appreciate even more the adaptive behaviors exhibited by the robots, the videos of all the 10 best bipeds and worms can be found respectively at https://youtu.be/-ECoa1tffok and https://youtu.be/jADw6Qf70g0. As a side comment, we found the gaits evolved with the self-organizing controller to be particularly fluid when compared to our previous works (Medvet, Bartoli, De Lorenzo, & Fidel, 2020; Medvet et al., 2021; Talamini et al., 2019).

Second, the evolved controllers do increase well beyond their initial size during evolution. As a matter of fact, the numbers reported in Table 1 imply there is a 149 % median increase in $|G|$ for the biped shape, and a 135 % median increase for the worm shape. Figure 3 corroborates this finding by showing an upward trend in the $|G|$ lines over the evolution (right plot), even though it appears to decelerate toward the end. At the same time, there does not seem to be an unexpected difference in contribution to $|G|$ between nodes and edges (the latter being slightly more abundant than the former).

By manually inspecting the evolved controllers, in particular the locations of nodes and edges over the body, we noticed some patterns. Most notably, long-

range edges do arise. For example, numerous edges connect the front part to the rear part of many worms. Intuitively, they establish a feedback mechanism that is necessary to inch forward as a caterpillar would do. Another recurring feedback mechanism evolved between the two legs of bipeds, contributing to their signature gait (i.e., hopping between the front and the rear leg). These facts altogether point out that self-organization is indeed at work.

5.1.1 Impact of topology optimization

By looking closer at Figure 3, it looks like velocity reaches a plateau well before controller growth stops. Indeed, there is no evidence controller growth stops at all. It might be the case that there exists some sort of "critical mass" of the controller such that, after a local optimum in the fitness space is reached, evolution keeps adding redundant genetic material (i.e., neurons and edges that do not necessarily contribute to the locomotion skills of the VSR). Since we do not disfavor in any way the addition of new edges and nodes that do not modify the functionality of the overall controller, we can speculate we are witnessing an instance of the bloat phenomenon (Silva & Costa, 2008) observed in the evolution of computer programs (also known as genetic programming, see Koza (1993)). It has been argued that bloat is beneficial to the individual as it provides a buffer against the deleterious effects of mutation and recombination (López et al., 2011). Interestingly, there is a rich body of literature in biology on the benefits of neutrality, after the work of Kimura (1979).

But then, does self-organization really matter? In other words, if (we speculate) bloat is at work, it might be that all those hidden neurons and edges are just excess genetic material, and evolution is simply optimizing the set of initial edges (those between sensors and actuators) and their parameters. To shed light on this argument, we performed an ablation study, aiming at exploring what happens if we evolve only the weights and biases and not the topology of the controller (which is stuck to be the initialization topology, see Section 4.4). In particular, we reset $p_{\text{mut}} = 1$ and all other mutation probabilities to 0. Figure 5 summarizes the results by showing the boxplots for the distribution of the $\bar{v}_x$ of the best individuals at the last generation for the two cases: with and without topology optimization. Interestingly, the distributions are radically different. We carried out a one-sided Mann Whitney U test with the null hypothesis that, for each shape, the median of the best fitness for evolving the topology is greater than the median best fitness for not evolving the topology. We found that the null hypothesis can be rejected at the 0.05 significance level (p-value < 0.001 for both shapes). Even though this result is valid only for this kind of initialization, it still suggests that self-organization does indeed benefit evolution.

5.2 RQ2: module transferability

Because of the way we defined the controller, specifically, since we put nodes inside voxels, it is clear that it can *by-design* be disassembled together with the morphology. Nevertheless, achieving module transferability in practice requires

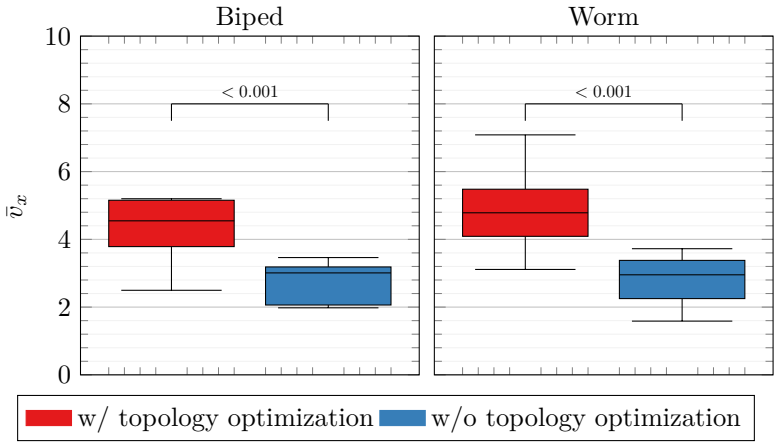


Figure 5: Boxplots for the distribution of the velocity $\bar{v}_x$ of the best individuals at the last generation, for biped (left) and worm (right), obtained with or without topology optimization. The bars above pairs of boxes show the corresponding p-value.

also a reassembly phase, in which disassembled modules are reused for building a new VSRs. We hence define a way of transferring modules, by means of a disassembly-reassembly procedure, and a way for measuring the effectiveness of the transfer.

5.2.1 Disassembly-reassembly procedure

Let r_1, r_2 be two *donor robots* that have been optimized for a given task. We assume that each of the robots can be disassembled in at least two modules, a *module* being a connected subset of robot voxels. The disassembly-reassembly procedure consists in building a new, reassembled robot r' by combining at least one module from the first robot and at least one module from the second robot. We measure the *transferability* of the two modules used to build r' as the relative ability of r' to solve the task with respect to the ability of the donors r_1, r_2. We remark that this definition of transferability is somehow limited, as it does not measure the degree to which robot modules can be re-used for arbitrarily different tasks. Nevertheless, we believe that the possibility of reusing modules for the same task is indeed useful (and, hence, it is important to quantify this possibility): in the long term, re-using parts of disposed robots might be an enabling factor for fully autonomous robotic ecosystems (Hale et al., 2019).

In the specific scenario of locomotion and with robots equipped with our controller, we cast this general definition as follows. When disassembling a robot, for each resulting module we drop all the edges whose source or target nodes are not located in voxels belonging to the module. When reassembling a robot from existing modules, we do not add any new edge between nodes

located in different modules—in some preliminary experiments, we explored other options, e.g., "rewiring" crossing edges at random, but we found they do not deliver any advantage. We measure transferability as:

$$\rho = \frac{\bar{v}_{x,r'}}{\frac{\bar{v}_{x,r_1} + \bar{v}_{x,r_2}}{2}}, \tag{4}$$

where $\bar{v}_{x,r_1}$, $\bar{v}_{x,r_1}$, and $\bar{v}_{x,r'}$ are the velocities of the three robots.

Since some edges of the controller get dropped in the disassembly-reassembly procedure, before measuring the velocity $\bar{v}_{x,r'}$ of the reassembled robot r', we re-optimized it by means of the same EA used for from-scratch optimization (see Algorithm 1), yet starting from a different initial population. This re-optimization is of crucial importance, since we want the new VSRs to be effective and we expect the reassembled robot to benefit from a reasonable amount of fine-tuning. The overall goal, however, is of making reassembly of pre-optimized modules cheaper than optimizing from scratch. In detail, the initial population of the re-optimization is composed of the controller G' of the reassembled robot r' and $n_{\text{pop}} - 1$ mutations of G' obtained by applying the same mutation operators (with the same probabilities) of the optimization step (see Section 4.4).

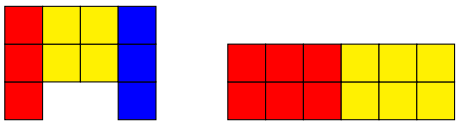
We remark that the disassembly-reassembly procedure requires to (a) define a partitioning of the two donor robots r_1 and r_2, (b) select one or more modules of r_1 and one or more modules of r_2, and (c) define a way to combine the selected modules. It is evident that modules transferability strongly depends on these three key choices. As an example, consider the case in which two "trunks" coming from two legged robots with different morphologies, each with a trunk and a few legs, are glued together: it is very unlikely that the resulting "trunk-only" robot will be effective in locomotion, since it has no legs. The representation of the controller on transferability, hence, clearly plays a secondary role. In the next sections, for answering RQ2, we manually choose reasonable options for these three choices. Later, in Section 5.3, we show how our representation may be helpful while coping with the first choice (partitioning a robot in modules) automatically.

5.2.2 Experimental procedure

We considered the two morphologies of the previous experiments and manually partitioned them in modules as shown in Figure 6. Then, we proceeded as follows.

First, we performed 10 evolutionary runs for each of the two morphologies, obtaining ten bipeds and ten worms. Then, we performed two types of reassembly for each morphology: an homogeneous one, where modules of the reassembled robots come from robots with the same morphology, and an heterogeneous one, where modules came from robots with the other morphology.

In the homogeneous case, we proceeded as follows. For the biped and for each $i \in \{1, \ldots, 10\}$, we reassembled a new biped by taking the red and blue modules (the "legs") from the i-th biped (i.e., the best individual obtained at the end of the i-th run) and the yellow module (the "torso") from the $((i + 1) \bmod 10)$-th



(a) Biped (b) Worm

Figure 6: Our manual partitioning in modules (one color for each module) of the two morphologies.

biped. Similarly, for the worm and for each $i \in \{1, \ldots, 10\}$, we reassembled a new worm by taking the red module (the "back half") from the i-th worm and the yellow module (the "front half") from the $((i + 1) \bmod 10)$-th worm.

In the heterogeneous case, we proceeded as follows. For the biped and for each $i \in \{1, \ldots, 10\}$, we reassembled a new biped by taking the red and blue modules from the i-th biped and the yellow module from the i-th worm. For the worm and for each $i \in \{1, \ldots, 10\}$, we reassembled a new worm by taking the red module from the i-th worm and the yellow module from the i-th biped.

After reassembly, we re-optimized each resulting reassembled robot with the EA of Algorithm 1 with the population initialization procedure modified as described in Section 5.2.1. For the re-optimization, we set $n_{\text{gen}} = 510$, a computational budget which is remarkably lower than the one used for optimization ($\approx 17\,\%$).

Upon re-optimization, we measure transferability ρ as defined in Equation (4). We also count how many edges were cut during disassembly, and compute the sum of their weights and biases in absolute value, as proxies of how much destructive that operation is. To make figures comparable, we cast these indexes as relative to their corresponding values before the disassembly. We denote by η_{num} the ratio between the number of edges dropped from a module upon disassembly and the overall number of edges in the module (before disassembly). Similarly, we denote by η_{weight} the ratio between the sum of weight and bias (in absolute value) of edges dropped from a module upon disassembly and the overall sum for edges in the module (before disassembly).

In the next subsection, we present the results and comment on them.

5.2.3 Results

Figure 7 reports the median (across the 10 reassembled robots for each morphology and each reassembly type, i.e., homogeneous and heterogeneous reassembly) ρ during the re-optimization for four combinations. We remark that a value of $\rho = 1.0$ (on the y-axis) corresponds to fully recovering the average velocity of the donors.

Our disassembly-reassembly procedure partially succeeds in recovering the lost functionality, in the homogeneous as well as in the more challenging heterogeneous combinations. Due to the intrinsic hardness of the heterogeneous reassembly case, and for the sake of brevity, in the following, we discuss only

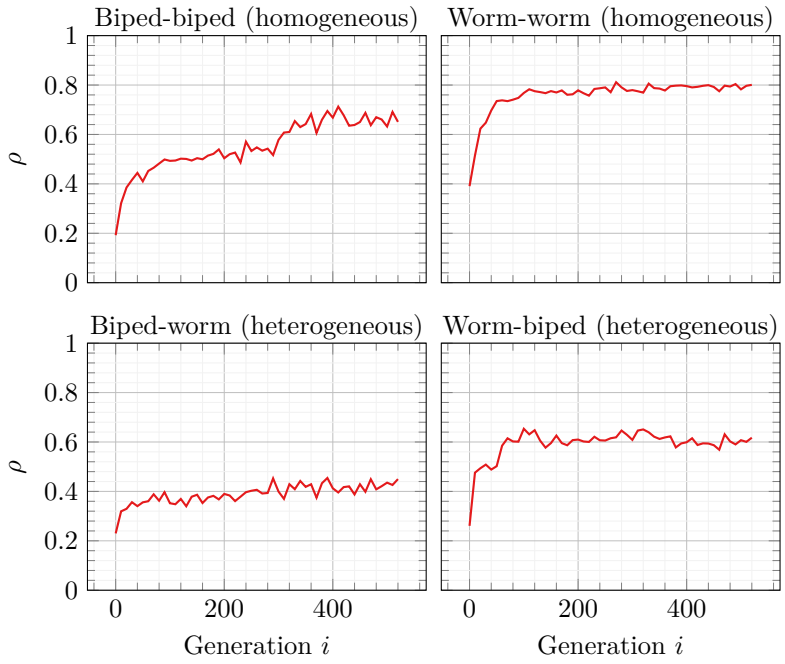


Figure 7: Median ρ, across the 10 reassembled robots for each morphology and reassembly type, during the re-optimization for four combinations.

the homogeneous case.

The magnitude of the effect differs by morphology; bipeds hover above 60 % and worms just fall short of 80 %. Nevertheless, we are far from recovering (let alone outclassing) the average velocity $\bar{v}_x$ of donors. Arguably, the morphology also affects transferability, with more "primitive" shapes like worm being advantaged.

For explaining why recovering is not complete, we looked at the values of η_{num} and η_{weight}, that measure the impact of disassembly in terms of edges being cut. Median values are $\eta_{\text{num}} = 0.60$ and $\eta_{\text{weight}} = 0.56$, for the biped, and 0.45 and 0.38 for the worm. In other words, the disassembly of a robot in modules is a very destructive operation for the VSR controller and the re-optimization struggles in recovering the lost structure. Interestingly, values of η_{num} and η_{weight} are lower for the worm (fewer edges are cut) and this is somehow reflected in the value of ρ, that is greater for the worm than for the biped.

5.2.4 Fostering modularity

Having observed that the removal of edges crossing modules appears to be detrimental to transferability, we designed a variant of the EA for evolving controllers that is aimed at discouraging the use of edges crossing modules. In other words, with this variant we can foster the modularity of the controller.

In detail, the EA is the same as the one presented in Section 4.4, with the exception of two mutation operators. We introduce a *biased edge addition* operator, and a *biased node addition* operator, that substitute the original edge addition and node addition operators (edge removal and node removal are left untouched, as well as the mutation probabilities).

In the biased edge addition operator, when picking the pair of nodes u, v to be connected by the new edge (see Section 4.4), instead on using uniform probability, we pick pairs whose nodes are in the same module with a probability that is ω larger than the probability of picking nodes that are in different modules.

In the biased node addition operator, choice of nodes u, v works as above; the new node w is placed in a random voxel of the module, if u, v are in the same module, or in a random voxel of the robot, otherwise.

The role of the parameter ω is to determine the degree of preference toward modularity. With $\omega > 1$ we foster modularity, with $\omega < 1$ we discourage modularity, with $\omega = 1$ we are neutral, i.e., we use exactly the same operators of the original EA of Section 4.4.

While designing this variant, i.e., the two biased mutation operators, we got inspiration from nature. In fact, G. P. Wagner et al. (2001) suggest that only two processes can foster modularity in biological systems: parcellation and integration. The latter consists in the selective acquisition of genes that map to phenotypic traits belonging to the same module. In other words, modular phenotypic traits are favored by evolution. In the light of this consideration, biased edge addition and biased node addition, by favoring connectivity within modules, can be seen as a simplified form of integration. The same processes (of parcellation and integration) have been fruitfully exploited for modular ANNs

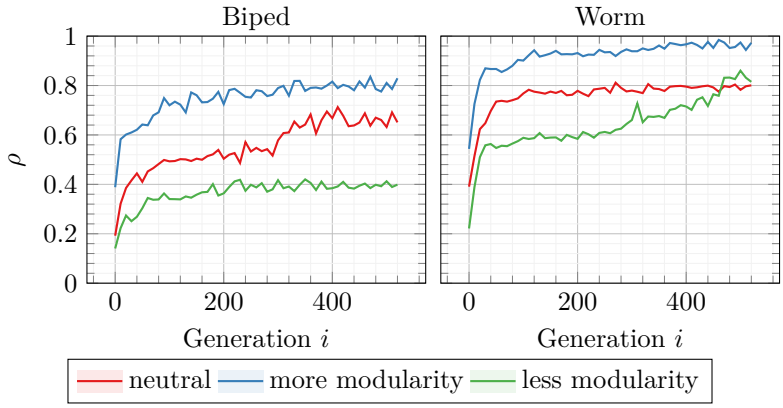


Figure 8: Median ρ, across the 10 reassembled robots for each morphology, during the re-optimization for biped (left) and worm (right) and for the three EA variants (line color).

in Mouret and Doncieux (2009), but dealing with a different set of tasks and not under an embodiment perspective.

For assessing the effectiveness of this variant in fostering modularity, i.e., favoring module transferability, we performed an experimental campaign following the same procedure described in Section 5.2.2 with three EAs: the original one ($\omega = 1$, "neutral" in the figures), one with $\omega = 10$ ("more modularity" in the figures), and one with $\omega = 0.1$ ("less modularity" in the figures), the latter as a sort of control group.

Figure 8 summarizes the main outcome of this experimental campaign by reporting the median (across the 10 reassembled robots for each morphology) ρ during the re-optimization for the two morphologies and the three variants.

The foremost observation is that the two new variants perform as expected—the result of the "neutral" variant is, obviously, the same as in the previous experiments. There seems to be a stark contrast between the median profile of ρ of the different variants. Not only do modular controllers, i.e., those obtained with $\omega = 10$, recover more quickly than neutral ones, i.e., those obtained with $\omega = 1$, but they also appear to be more fit at the very end of evolution. The same holds true when comparing neutral controllers with their non-modular counterparts, i.e., those obtained with $\omega = 0.1$. Statistical tests comparing median ρ across the evolutionary runs showed the differences at the end of re-optimization are indeed significant, with the exception of the neutral vs. modular and neutral vs. non-modular pairings for the worm shape.

We hypothesize that, intuitively, the observed differences are due to the different number of edges that are cut upon disassembly for the three variants, that we report in Figure 9. It can be seen from the figure that modular controllers stay in the range of 0.25–0.5 of edges cut (η_{num}), while the proportions are higher for the neutral and non-modular variants. The plots for η_{weight} cor-

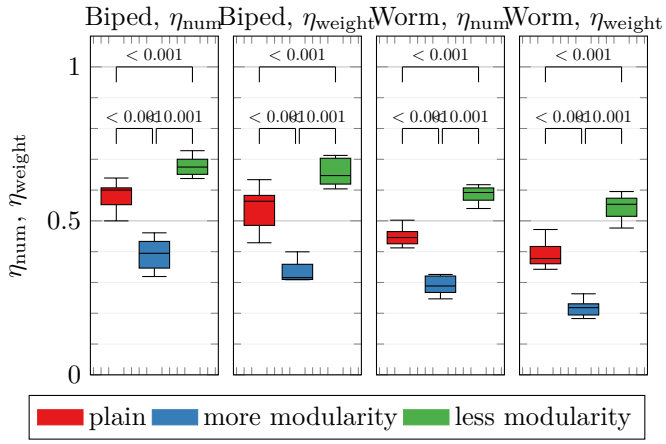


Figure 9: Boxplots for the distribution of η_{num} of η_{weight} (rate of edges cut upon disassembly) for the two morphologies and the three EA variants. The bars above pairs of boxes show the corresponding p-value.

roborate this interpretation. Statistical tests, whose outcomes are reported in Figure 9, support our claim that the three variants are different in terms of η_{num} and η_{weight}.

As an aside, by looking at the raw values of $\bar{v}_x$ after the re-optimization, we observe that the magnitude of the loss with respect to the value of the donor robots varies with the morphology, being less pronounced for worms, and more evident for bipeds, and that worms have, on average, less edges cut. Undoubtedly, the configuration influences how large is the gap between bipeds and worms, with modular biped controllers being more or less at the same level as their worm counterparts, whereas we witness a 30% hiatus with non-modular controllers. For this, we conjecture the reason to be that the long-range edges (non-modular controllers are biased toward) bear more importance in bipeds rather than worms. This fact is meaningful if we consider the gait of bipeds and the many long-range edges that evolve between the two legs (see Section 5.1).

For gaining further insights about the three variants, we compared them in terms of the outcome of the first optimization, i.e., we looked at the values of $\bar{v}_x$ and $|G|$ obtained before the disassembly-reassembly procedure. Figures 10 and 11 summarizes the results concerning these indexes, respectively as the median values for $\bar{v}_x$ and $|G|$ during the evolution and as the boxplots for the distributions of the value of $\bar{v}_x$ of the best individuals at the end of the evolution.

It can be seen that, for what concerns the size $|G|$ of controllers, the three variants exhibit negligible differences. From another point of view, favoring or discouraging modularity does not impact on the self-organization of the controller.

Concerning the velocity $\bar{v}_x$, Figure 10 seems to suggest that there are some differences. In particular, it looks like modular evolved VSRs are, on average,

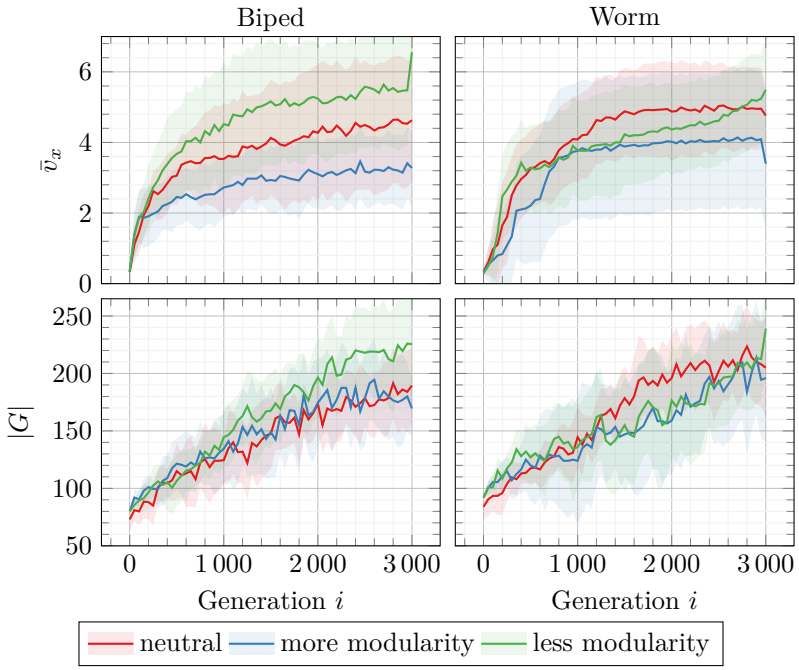


Figure 10: Median ± standard deviation (solid line and shaded area) across the 10 runs of the robot velocity $\bar{v}_x$ (top) and controller size $|G|$ (bottom) for the best individual, for biped (left) and worm (right) and for the three EA variants.

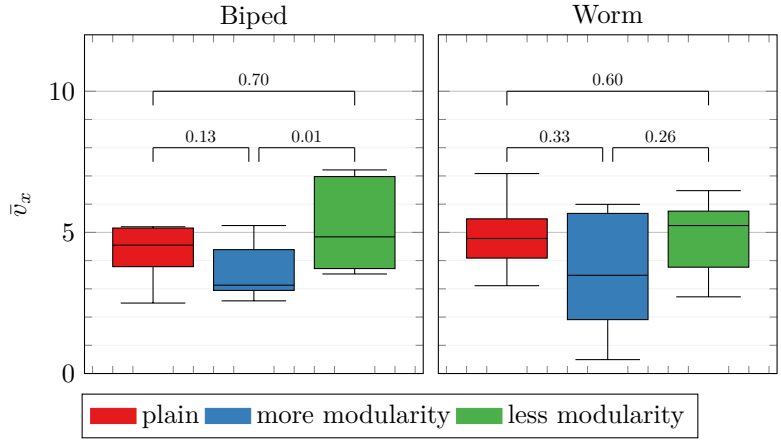


Figure 11: Boxplots for the distribution of the velocity $\bar{v}_x$ of the best individuals at the last generation, for biped (left) and worm (right), obtained with the three EA variants. The bars above pairs of boxes show the corresponding p-value.

less fit than neutral ones, which in turn seem to be, on average, less fit than their non-modular counterparts. Despite those differences are not statistically significant (see the p-values in Figure 11), we think they can be explained by the role of long-range edges, i.e., edges crossing several voxels. Non-modular controllers might outperform the others because their long-range edges (they are implicitly biased toward) carry "more" information. More broadly, the effectiveness of a procedure for favoring or disfavoring modularity, might depend on the choice on how to split the robot in modules. However, we here compared EA variants using the same choice for modules and we do not have any argument for hypothesizing that a different choice might be better for favoring or disfavoring modularity.

To summarize, two conclusions can be made. First, in line with the embodied cognition paradigm, morphology impacts transferability. Second, after disassembly-reassembly, we do not always fully recover the lost functionality of the donor robots. In particular, the degree of modularity does impact the transferability of an embodied controller. There are, however, two caveats: (a) modularity appears to negatively affect average velocity $\bar{v}_x$ before disassembly and (b) favoring modularity requires the manual definition of modules before the optimization. The latter point is undeniable, but we will next introduce a procedure for the automatic discovery of modules inside VSRs having an embodied, self-organizing controller.

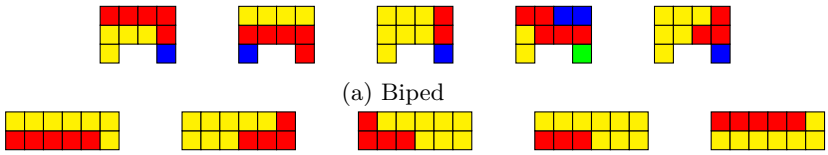
5.3 RQ3: automatic modules discovery

So far, we assumed that the partitioning in modules of a given morphology was known *a priori*, i.e., before the optimization of a controller for that morphology. We were hence considering a *human-in-the-loop* scenario, in which the human designer plays a key role in defining modules.

In this section, we propose a way for discovering modules *a posteriori*, i.e., after the optimization, and automatically: we hence consider a *human-out-of-the-loop* scenario. More specifically, we propose a method to partition a VSR with a given morphology into modules, inclusive of both body and brain, after the optimization of an embodied, self-organizing controller for that morphology.

While the human-in-the-loop setting is interesting per se, e.g., for transferring robotic components and assembling new robots, the human-out-the-loop would be interesting even in the long-term, visionary setting of an autonomous robotic ecosystem; in this scenario, the ecosystem could be made more efficient by reusing (recycling) components of robots that have to be disposed of. As principled instantiation of this scenario, consider the ARE project (Hale et al., 2019).

The proposed method works as follows. Let r be the VSR to be partitioned in modules and let B be the morphology (i.e., the set of voxels with their positions in the grid) of r and $G = (V, E)$ the controller of r. Let (B', G') be a *candidate module*, with $B' \subseteq B$ being a connected subset of voxels of r and $G' = (V', E')$ the subgraph of G containing all the vertexes $V' \subseteq V$ located in voxels of B' and all the edges $E' \subseteq E$ whose source or target nodes are in V'. We define the



(a) Biped

(b) Worm

Figure 12: A random subset of robots partitioned in modules with the automatic procedure, one color for each module. The module with the greatest compactness is depicted in red.

module compactness $c(B', G')$ of a candidate module (B', G') as:

$$c(B', G') = \frac{\sum e \in E'_{\text{inside}} |e.\text{weight}| + |e.\text{bias}|}{\sum e \in E' |e.\text{weight}| + |e.\text{bias}|}, \tag{5}$$

where $E'_{\text{inside}} = \{e \in E' : e.\text{source} \in V' \wedge e.\text{target} \in V'\}$ is the subset of E' edges which do not cross the boundaries of the module, i.e., whose source or target nodes are in V'.

Based on the module compactness definition, we propose this procedure for partitioning a VSR into modules. First, we identify all the candidates modules whose size $|B'|$ is in a given range $[b_{\min}, b_{\max}]$. Second, we find the candidate module with the greatest compactness and partition the robot in that module and the modules remaining after removing it. The number of modules resulting from this procedure can be larger than 2, if, upon the removal of the module with the greatest compactness, the remaining part of the robot is not physically connected—see, for example, Figure 12a.

The rationale of this procedure is to exploit the findings of the previous experiments, that suggest that cutting edges is deleterious with respect to module transferability. We remark that the procedure itself is applicable because the controller is a graph distributed over the body.

Other graph clustering techniques exist in the literature that we could have used in place of module compactness. To name a few, these include minimum cut algorithms (Goldschmidt & Hochbaum, 1988), spectral clustering (Seary & Richards, 1996), and community detection algorithms (Girvan & Newman, 2001). While any of the above would be a legitimate choice, we here adopt a simple approach that is still intuitive, easy to implement and effective.

We performed a qualitative evaluation of this procedure by applying it to the VSRs evolved in the experiments described in Section 5.1. After preliminary experiments and taking into account the size of the biped and of the worm, we set $b_{\min} = 2$ and $b_{\max} = 5$. Figure 12 shows a few of the partitioning that we obtained in this experiment.

By looking at the modules with greatest compactness (i.e., the red ones), we notice some patterns. For the biped shape, there seems to be a tendency to find modules that span horizontally across the body, connecting the front with the rear. It might be the case that dense neural connectivity subsists between

these two body parts. This is likely to be useful in a gait that is alternated like the biped one. To a lesser extent, the same considerations can be made for the worm shape.

We envision such modules to be of high utility when assembling brand new VSRs from pre-optimized components. As a result, this heuristic could well fit in an automatic pipeline of robot disassembly and assembly, where new VSRs are fabricated in a human-out-of-the-loop manner by assembling modules picked from a repository and briefly fine-tuning them.

6 Concluding remarks

We have considered the case of VSRs, a kind of robots that are intrinsically modular in the morphology. Existing methods for building controllers for VSRs are not, however, capable of exploiting the modularity of the morphology. Addressing this limitation would permit to disassemble robots in modules, encompassing both the body and the brain, and to reassemble them differently, to cope, e.g., with malfunctions, broken components, or different environments and tasks. In the long term, enhancing modularity of VSRs would enable the building of libraries of pre-optimized modules that can be reused over and over.

We proposed a representation for an embodied, self-organizing neural controller in which nodes and edges of the ANN are located at precise voxels in the VSR morphology. We also described an EA suitable for evolving a controller with our representation given a task and a morphology. With an extensive experimental campaign, we showed that:

(i) our representation and EA allow to obtain effective controllers for the task of locomotion;

(ii) VSRs with evolved controllers can be disassembled and reassembled in different VSRs and are able, after a cheap re-optimization, to recover the original functionality;

(iii) modularity can be favored or discouraged by means of a simple numerical parameter in the mutation operators of the EA and this is reflected in the abilitity of recovering the functionality;

(iv) due to its self-organizing property, our controller can be helpful for partitioning automatically a VSRs in modules that exhibit potentially good transferability.

7 Data and code policy

We made all the code to replicate the experiments publicly available at https://github.com/pigozzif/SelfOrganizingVSRController.

Acknowledgments

F.P. was partially supported by a Google Faculty Research Award granted to E.M.. The experimental evaluation of this work was done on CINECA HPC cluster within the CINECA-University of Trieste agreement. The authors wish to thank the anonymous reviewers for their helpful comments.

References

Baguna, J., Saló, E., & Auladell, C. (1989). Regeneration and pattern formation in planarians. iii. that neoblasts are totipotent stem cells and the cells. *Development*, *107*(1), 77–86.

Bloom, P. (2004). *Descartes' baby: How the science of child development explains what makes us human*.

Bongard, J. (2011). Spontaneous evolution of structural modularity in robot neural network controllers: Artificial life/robotics/evolvable hardware. *GECCO '11*.

Bongard, J. C., Bernatskiy, A., Livingston, K. R., Livingston, N., Jr., J. H. L., & Smith, M. L. (2015). Evolving robot morphology facilitates the evolution of neural modularity and evolvability. In S. Silva & A. I. Esparcia-Alcázar (Eds.), *Proceedings of the genetic and evolutionary computation conference, GECCO 2015, madrid, spain, july 11-15, 2015* (pp. 129–136). ACM. https://doi.org/10.1145/2739480.2754750

Brooks, R. A. (1990). Elephants don't play chess [Designing Autonomous Agents]. *Robotics and Autonomous Systems*, *6*(1), 3–15. https://doi.org/https://doi.org/10.1016/S0921-8890(05)80025-9

Buchanan, E., Le Goff, L. K., Li, W., Hart, E., Eiben, A. E., De Carlo, M., Winfield, A. F., Hale, M. F., Woolley, R., Angus, M., et al. (2020). Bootstrapping artificial evolution to design robots for autonomous fabrication. *Robotics*, *9*(4), 106.

Cappelle, C., Bernatskiy, A., Livingston, K., Livingston, N., & Bongard, J. (2016). Morphological modularity can enable the evolution of robot behavior to scale linearly with the number of environmental features. *Frontiers Robotics AI*, *3*, 59.

Cheney, N., Clune, J., & Lipson, H. (2014). Evolved electrophysiological soft robots. *Artificial Life Conference Proceedings 14*, 222–229.

Cheney, N., Bongard, J., & Lipson, H. (2015). Evolving soft robots in tight spaces. *Proceedings of the 2015 annual conference on Genetic and Evolutionary Computation*, 935–942.

Cheney, N., MacCurdy, R., Clune, J., & Lipson, H. (2013). Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding. *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, 167–174.

Clune, J., Beckmann, B. E., McKinley, P. K., & Ofria, C. (2010). Investigating whether hyperneat produces modular neural networks. *Proceedings of*

the 12th annual conference on Genetic and evolutionary computation, 635–642.

Clune, J., Mouret, J.-B., & Lipson, H. (2013). The evolutionary origins of modularity. *Proceedings. Biological sciences / The Royal Society*, *280*, 20122863. https://doi.org/10.1098/rspb.2012.2863

Corucci, F., Cheney, N., Giorgio-Serchi, F., Bongard, J., & Laschi, C. (2018). Evolving soft locomotion in aquatic and terrestrial environments: Effects of material properties and environmental transitions. *Soft robotics*, *5*(4), 475–495.

De Jong, K. A. (2006). *Evolutionary Computation: A Unified Approach*. MIT Press.

de Garis, D. H. (1998). Genetic programming - building artificial nervous systems with genetically programmed neural network modules. *Proceedings of the 7th International Conference on Machine Learning*.

Eiben, A. (2021). Real-world robot evolution: Why would it (not) work? *Frontiers in Robotics and AI*, *8*, 243. https://doi.org/10.3389/frobt.2021.696452

Ellefsen, K. O., Mouret, J.-B., & Clune, J. (2015). Neural modularity helps organisms evolve to learn new skills without forgetting old skills. *PLoS Computational Biology*, *11*.

Espinosa-Soto, C., & Wagner, A. (2010). Specialization can drive the evolution of modularity. *PLOS Computational Biology*, *6*(3), 1–10. https://doi.org/10.1371/journal.pcbi.1000719

Faiña, A., Bellas, F., Orjales, F., Souto, D., & Duro, R. (2015). An evolution friendly modular architecture to produce feasible robots. *Robotics Auton. Syst.*, *63*, 195–205.

Faiña, A. (2021). Evolving modular robots: Challenges and opportunities. *ALIFE 2021: The 2021 Conference on Artificial Life*.

Ferigo, A., Iacca, G., & Medvet, E. (2021). Beyond Body Shape and Brain: Evolving the Sensory Apparatus of Voxel-based Soft Robots. *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*.

Ferigo, A., Iacca, G., Medvet, E., & Pigozzi, F. (2021). Evolving hebbian learning rules in voxel-based soft robots.

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

Gentile, L., Cebrià, F., & Bartscherer, K. (2011). The planarian flatworm: An in vivo model for stem cell biology and nervous system regeneration. *Disease Models & Mechanisms*, *4*, 12–19.

Girvan, M., & Newman, M. (2001). Community structure in social and biological networks. *proc natl acad sci*, *99*, 7821–7826.

Goldschmidt, O., & Hochbaum, D. (1988). Polynomial algorithm for the k-cut problem. *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, 444–451.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning* [http://www.deeplearningbook.org]. MIT Press.

Gould, S. J., & Lewontin, R. C. (1979). The spandrels of san marco and the panglossian paradigm: A critique of the adaptationist programme. *Proceedings of the Royal Society of London Series B, Biological Sciences*, 581–598.

Gutai, A., & Gorochowski, T. (2021). How biological concepts and evolutionary theories are inspiring advances in machine intelligence. *Preprints*. https://doi.org/10.20944/preprints202109.0234.v1

Hale, M., Buchanan, E., Winfield, A., Timmis, J., Hart, E., Eiben, A., Angus, M., Veenstra, F., Li, W., Woolley, R., De Carlo, M., & Tyrrell, A. (2019). The are robot fabricator: How to (re)produce robots that can evolve in the real world. *International Society for Artificial Life: ALIFE2019*, 95–102. https://doi.org/10.1162/isal_a_00147

Hiller, J., & Lipson, H. (2012). Automatic design and manufacture of soft robots. *IEEE Transactions on Robotics*, *28*(2), 457–466.

Howison, T., Hauser, S., Hughes, J., & Iida, F. (2021). Reality-assisted evolution of soft robots through large-scale physical experimentation: A review. *Artificial Life*, *26*, 484–506. https://doi.org/10.1162/artl_a_00330

Joachimczak, M., Suzuki, R., & Arita, T. (2016). Artificial metamorphosis: Evolutionary design of transforming, soft-bodied robots. *Artificial Life*, *22*(3), 271–298.

Johnson, B. R., & Lam, S. (2010). Self-organization, natural selection, and evolution: Cellular hardware and genetic software.

Joven, A., Elewa, A., & Simon, A. (2019). Model systems for regeneration: Salamanders. *Development*, *146*.

Kamimura, A., Kurokawa, H., Yoshida, E., Tomita, K., Kokaji, S., & Murata, S. (2004). Distributed adaptive locomotion by a modular robotic system, m-tran ii. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, *3*, 2370–2377 vol.3. https://doi.org/10.1109/IROS.2004.1389763

Kimura, M. (1979). The neutral theory of molecular evolution. *Scientific American*, *241 5*, 98–100, 102, 108 passim.

Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex Syst.*, *4*.

Koza, J. (1993). Genetic programming - on the programming of computers by means of natural selection. *Complex adaptive systems*.

Kriegman, S., Blackiston, D., Levin, M., & Bongard, J. (2020). A scalable pipeline for designing reconfigurable organisms. *Proceedings of the National Academy of Sciences*, *117*(4), 1853–1859.

Kriegman, S., Nasab, A. M., Shah, D., Steele, H., Branin, G., Levin, M., Bongard, J., & Kramer-Bottiglio, R. (2020). Scalable sim-to-real transfer of soft robot designs. *2020 3rd IEEE International Conference on Soft Robotics (RoboSoft)*, 359–366. https://doi.org/10.1109/RoboSoft48309.2020.9116004

Lawrence, J. (2020). Arm loss and regeneration in asteroidea (echinodermata). https://doi.org/10.1201/9781003077572-7

Lipson, H., SunSpiral, V., Bongard, J., & Cheney, N. (2016). On the difficulty of co-optimizing morphology and control in evolved virtual creatures. *Artificial Life*, 226–233.

Liu, J., Zhang, X., & Hao, G. (2016). Survey on research and development of reconfigurable modular robots. *Advances in Mechanical Engineering*, *8*.

López, E., Poli, R., Kattan, A., O'Neill, M., & Brabazon, A. (2011). Neutrality in evolutionary algorithms… what do we know? *Evolving Systems*, *2*, 145–163.

Mann, H. B., & Whitney, D. (1947). On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, *18*, 50–60.

Medvet, E., & Bartoli, A. (2020). Evolutionary optimization of graphs with graphea. *International Conference of the Italian Association for Artificial Intelligence*, 83–98.

Medvet, E., Bartoli, A., De Lorenzo, A., & Fidel, G. (2020). Evolution of distributed neural controllers for voxel-based soft robots. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 112–120.

Medvet, E., Bartoli, A., De Lorenzo, A., & Seriani, S. (2020a). 2D-VSR-Sim: A simulation tool for the optimization of 2-D voxel-based soft robots. *SoftwareX*, *12*.

Medvet, E., Bartoli, A., De Lorenzo, A., & Seriani, S. (2020b). Design, Validation, and Case Studies of 2D-VSR-Sim, an Optimization-friendly Simulator of 2-D Voxel-based Soft Robots. *arXiv*, arXiv–2001.

Medvet, E., Bartoli, A., Pigozzi, F., & Rochelli, M. (2021). Biodiversity in evolved voxel-based soft robots. *Proceedings of the genetic and evolutionary computation conference* (pp. 129–137). Association for Computing Machinery. https://doi.org/10.1145/3449639.3459315

Miller, J. F., & Harding, S. L. (2008). Cartesian genetic programming. *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*, 2701–2726.

Mitchell, M. (2021). Why ai is harder than we think. *Proceedings of the Genetic and Evolutionary Computation Conference*, 3. https://doi.org/10.1145/3449639.3465421

Moreno, R., Veenstra, F., Silvera, D., Franco, J., Gracia, O., Cordoba, E., Gomez, J., & Faina, A. (2018). Automated reconfiguration of modular robots using robot manipulators. *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, 884–891. https://doi.org/10.1109/SSCI.2018.8628628

Mouret, J.-B., & Doncieux, S. (2009). Evolving modular neural-networks through exaptation. *2009 IEEE Congress on Evolutionary Computation*, 1570–1577.

Neumann, J. V., & Burks, A. W. (1966). *Theory of self-reproducing automata*. University of Illinois Press.

Nolfi, S., & Floreano, D. (2000). *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press.

Pfeifer, R., & Bongard, J. (2006). *How the body shapes the way we think: A new view of intelligence*. MIT press.

Rothlauf, F. (2006). Representations for genetic and evolutionary algorithms. *Representations for genetic and evolutionary algorithms* (pp. 9–32). Springer.

Rothlauf, F., & Goldberg, D. E. (2003). Redundant representations in evolutionary computation. *Evolutionary Computation*, *11*(4), 381–415.

Roy, S., & Gatien, S. (2008). Regeneration in axolotls: A model to aim for! *Experimental gerontology*, *43*, 968–73. https://doi.org/10.1016/j.exger.2008.09.003

Rus, D., & Tolley, M. T. (2015). Design, fabrication and control of soft robots. *Nature*, *521*(7553), 467.

Schäfer, A. M., & Zimmermann, H. G. (2006). Recurrent neural networks are universal approximators. *International Conference on Artificial Neural Networks*, 632–640.

Schrum, J., & Miikkulainen, R. (2014). Evolving multimodal behavior with modular neural networks in ms. pac-man. *GECCO 2014 - Proceedings of the 2014 Genetic and Evolutionary Computation Conference*. https://doi.org/10.1145/2576768.2598234

Seary, A., & Richards, W. (1996). Partitioning networks by eigenvectors. *Proc. Int. Conf. on Social Networks*, *1*.

Segev, I., & Schneidman, E. (1999). Axons as computing devices: Basic insights gained from models. *Journal of Physiology-Paris*, *93*, 263–270.

Silva, S., & Costa, E. (2008). Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. *Genetic Programming and Evolvable Machines*, *10*, 141–179.

Sims, K. (1994). Evolving 3d morphology and behavior by competition. *Artificial Life*, *1*, 353–372. https://doi.org/10.1162/artl.1994.1.4.353

Stanley, K., D'Ambrosio, D., & Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, *15*, 185–212. https://doi.org/10.1162/artl.2009.15.2.15202

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, *10*(2), 99–127.

Such, F., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., & Clune, J. (2017). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *ArXiv*, *abs/1712.06567*.

Sui, X., Cai, H., Bie, D., Zhang, Y., Zhao, J., & Zhu, Y. (2020). Automatic generation of locomotion patterns for soft modular reconfigurable robots. *Applied Sciences*, *10*(1), 294.

Talamini, J., Medvet, E., Bartoli, A., & De Lorenzo, A. (2019). Evolutionary synthesis of sensing controllers for voxel-based soft robots. *Artificial Life Conference Proceedings*, 574–581.

Wagner, G., Pavlicev, M., & Cheverud, J. (2007). The road to modularity. *Nature Reviews Genetics*, *8*, 921–931.

Wagner, G. P., & Altenberg, L. (2005). Complex adaptations and the evolution of evolvability.

Wagner, G. P., Mezey, J., & Calabretta, R. (2001). *Natural selection and the origin of modules*. na.

Whitley, L. D. (1991). Fundamental principles of deception in genetic search. *Foundations of genetic algorithms* (pp. 221–241). Elsevier.

Yamashita, Y., & Tani, J. (2008). Emergence of functional hierarchy in a multiple timescale neural network model: A humanoid robot experiment. *PLoS computational biology*, *4*, e1000220. https://doi.org/10.1371/journal.pcbi.1000220

Yim, M., Shen, W.-M., Salemi, B., Rus, D., Moll, M., Lipson, H., Klavins, E., & Chirikjian, G. S. (2007). Modular self-reconfigurable robot systems [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, *14*(1), 43–52.