Does Aligning Phenotypic and Genotypic Modularity Improve the Evolution of Neural Networks?

Joost Huizinga, Jean-Baptiste Mouret, Jeff Clune

Does Aligning Phenotypic and Genotypic Modularity Improve the Evolution of Neural Networks?

Joost Huizinga
Evolving AI Lab
Dept. of Computer Science
University of Wyoming
jhuizing@uwyo.edu

Jean-Baptiste Mouret
Inria Nancy - Grand Est
CNRS UMR 7503
University of Lorraine
Villers-lès-Nancy, France
jean-baptiste.mouret@inria.fr

Jeff Clune
Evolving AI Lab
Dept. of Computer Science
University of Wyoming
jeffclune@uwyo.edu

ABSTRACT

Many argue that to evolve artificial intelligence that rivals that of natural animals, we need to evolve neural networks that are *structurally organized* in that they exhibit *modularity*, *regularity*, and *hierarchy*. It was recently shown that a cost for network connections, which encourages the evolution of modularity, can be combined with an indirect encoding, which encourages the evolution of regularity, to evolve networks that are both modular and regular. However, the bias towards regularity from indirect encodings may prevent evolution from independently optimizing different modules to perform different functions, unless modularity in the phenotype is *aligned* with modularity in the genotype. We test this hypothesis on two multi-modal problems—a pattern recognition task and a robotics task—that each require *different* phenotypic modules. In general, we find that performance is improved only when genotypic and phenotypic modularity are encouraged simultaneously, though the role of alignment remains unclear. In addition, intuitive manual decompositions fail to provide the performance benefits of automatic methods on the more challenging robotics problem, emphasizing the importance of automatic, rather than manual, decomposition methods. These results suggest encouraging modularity in both the genotype and phenotype as an important step towards solving large-scale multi-modal problems, but also indicate that more research is required before we can evolve structurally organized networks to solve tasks that require multiple, different neural modules.

Keywords

Artificial Neural Networks; Modularity; Regularity; HyperNEAT; NSGA-II

1. INTRODUCTION

A long term goal in evolutionary robotics is to evolve artificial neural networks with the intelligence, flexibility, and robustness of natural brains. To increase behavioral complexity, it is widely believed that these networks need to be *structurally organized* in that they should exhibit *modularity*, *regularity*, and *hierarchy* [20, 32].

In networks, a module refers to a community of nodes that are strongly connected to each other, while only sparsely connected to nodes in other communities [7, 16, 20]. Furthermore, those modules are considered functional if they perform a specific task or function [20]. Modularity provides evolutionary robustness and increased evolvability, as it allows mutations to locally affect one module, while leaving other modules mostly unchanged [5, 7, 16, 17, 29]. Modularity does not naturally evolve in artificial neural networks, but has been shown to emerge in a variety of domains and types of neural networks when a cost for connections is added [7, 13, 15, 21].

Regularity refers to the compressibility of the description of a structure [20]. That is, if a structure exhibits some form of regularity, such as repetition, symmetry, or self-similarity, it is sufficient to describe only part of the structure and how that part is reused. Reuse of information can increase both performance and evolvability on a variety of tasks by allowing for coordinated changes to the phenotype [6, 8, 14], but this reuse of information is only possible when the evolved structures are indirectly encoded.

Previous work has shown how to evolve neural networks that are both modular and regular by combining a connection cost with an indirect encoding [15]. However, this work focused on problems with repeated or symmetric subproblems. That raises the question of whether evolution, while optimizing a modular yet indirectly encoded structure, can optimize the evolved modules separately. Optimization of separate phenotypic modules may be facilitated by *aligned* genotypic modularity; which implies not only that the genotype is modular, but also that changes to genotypic modules only affect corresponding modules in the phenotype.

Modularity, both phenotypic and genotypic, has received considerable attention in recent publications [1, 5, 7, 9, 16, 17, 22, 29]. Analyses of biological brain networks reveal modular structures that would not be expected if these networks were randomly connected [4, 29]. Similarly, studies of gene regulatory networks find modular dynamics as well, where groups of genes are consistently coexpressed [9].

It has been suggested that, when studying the evolution of modularity in indirectly encoded networks, it is important to not only examine the structures of the phenotype and genotype, but also their alignment [1, 22]. If the phenotype is modular, but the genotype is not, it is unlikely that the phenotypic modules can be optimized separately, because changes to the genotype will affect the phenotype as a whole. Conversely, when the genotype is modular, but the phenotype is not, even if the modular genotype leads to local changes, there is a high probability that these changes still affect the performance of the individual as a whole. Even if both the genotype and the phenotype are modular, single mutations may still affect performance holistically if the genotypic modules do not *align* with the phenotypic mod-

ules, such that changes to a single genotypic module affect several or all phenotypic modules.

We investigate these issues by examining what happens when we encourage modularity in both the phenotype and the genotype on two multi-modal problems—a pattern recognition task and a robotics task—where the sub-problems are *different*. We examine whether evolution can not only form different modules, but also if it structures the genotype such that phenotypic modules can be optimized separately. We encourage modularity in two different ways: by adding a cost for network connections and by directly selecting for increased modularity. We compare these results with hand-crafted network architectures and find that performance generally increases when both the genotype and phenotype are modular, though the role of alignment remains unclear.

2. METHODS

2.1 HyperNEAT

HyperNEAT is an evolutionary algorithm for evolving artificial neural networks (ANNs) with an indirect encoding [31]. In HyperNEAT, the ANN is encoded by a Compositional Pattern Producing Network (CPPN), which is effectively a feed-forward neural network where nodes can have a range of different activation functions [30]. In the phenotype, every neuron has a geometric coordinate (Figs. 3 and 4). To determine the strength of a connection, the coordinates of its source and target neurons are provided to the CPPN, and the output of the CPPN determines the strength of the connection. While HyperNEAT was developed as a complete evolutionary algorithm, in this paper, it only refers to the encoding; our EA is described in section 2.3.

CPPNs were originally presented as having a single output node, the weight output, but our experiments also require a number of additional CPPN outputs: a bias output, a Link Expression Output (LEO) [33], and a time-constant output. The bias output determines the biases of the neurons in the ANN. When determining the bias of a neuron, we only have the coordinate for a single neuron, but our CPPN requires two sets of coordinate inputs. In these cases we set the coordinates for the second neuron to be all zeros. LEO determines whether a connection should be expressed or not, and has been shown to assist HyperNEAT in solving modular problems [33]. It is queried for every possible connection, and a connection is only expressed if LEO produces a value > 0. The time-constant output is only required in the robotics experiment, and it determines the time-constants of the neurons in the ANN (see section 2.2). CPPN outputs can only generate values in $[-1, 1]$, but their output is scaled to match the range of the parameter they describe.

To improve HyperNEAT's performance on robotics tasks, it is customary to have the geometric coordinates of the input and output neurons in the ANN reflect the physical coordinates of the sensors and actuators they are associated with. However, doing so can be problematic when sensors or actuators inhabit the exact same position on the robot. In those cases it can be helpful to represent these neurons as residing in a separate 'space'; an idea that was implemented with the multi-spatial substrate (MSS) [28]. With MSS, every neuron in the ANN is assigned to a 'space', and a CPPN output is added for every pair of 'spaces' that can be connected (including when a space can be connected to itself). To query the weight of a connection, its value has to be read from the CPPN output that is associated with the pair of spaces defined by the source and target neurons.

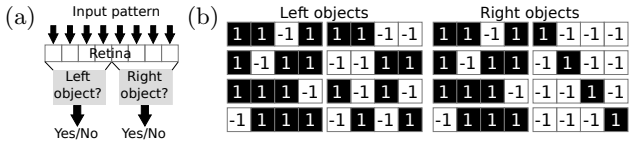


Figure 1: Multi-modal retina problem. (a) The left sub-task is independent from the right sub-task. (b) Half of the left objects are different from the right objects.

We have extended MSS to support CPPNs with more than one output by adding a complete *set* of outputs for every space (Figs. 3f and 4e, f). In addition, to better align or disalign phenotypic and genotypic decompositions, we do not add CPPN outputs for every *pair* of spaces, but only for every *individual* space. We then assign both ANN connections and ANN neurons to a space, giving us full control over which CPPN output governs which ANN element.

Our ANNs do not actually have nodes residing at the same coordinates, meaning that MSS is not necessary (Fig. 4a). However, preliminary experiments without MSS resulted in poor performance for all treatments, thus we did not explore this direction any further (data not shown).

2.2 Test Problems

We tested our hypothesis on two different problems: a pattern recognition problem and a robotics problem. The pattern recognition task is a variant of the retina problem [16]. In this variant, the ANN has 8 inputs and 2 outputs (Fig. 1a), but it is otherwise the same as described in [15]. The task of the ANN is to produce a value > 0 on the left output if the input-pattern for the first 4 inputs is a left object (Fig. 1b), and a value < 0 otherwise. Similarly, the right output should produce a value > 0 if the input-pattern for the last 4 inputs is a right object (Fig. 1b), and a value < 0 otherwise. An output value of 0 is always incorrect. If e is the number of errors produced by the network per output on all 256 input patterns, performance is calculated as $p = 1 - e/512$. In contrast to previous incarnations of the retina problem [15, 16, 33], in this paper the left and right objects are not mirror images of each other.

In the robotics task, the ANN evolves to control a 6-legged, radially symmetrical robot (Fig. 2a) simulated in and provided by the Bullet physics engine (version 2.81, in the Dynamic Control Demo), controlled as described in [27]. The robot has to perform 6 different tasks, each associated with one of its 6 inputs: move-forward, move-backward, turn-left, turn-right, jump, and crouch (Fig. 2b). The ANN is a CTRNN [3], with weights and biases in $[-2, 2]$, and time constants in $[1, 6]$. Activation functions are $f(x) = tanh(5x)$ for output neurons, and $f(x) = (tanh(5x) + 1)/2$ for hidden neurons.

Performance is evaluated in six sessions, one for each sub-task. At the start of each session the ANN is reset, the relevant task input is set to 1, and the robot is moved to $[0, 1, 0]$. Performance on each sub-task is defined as follows. Let $\vec{c}_t = [x_t, y_t, z_t]$ be the center of mass of the robot at time-step t, and let T be the total number of time-steps of a session. Forward (p_f), backward (p_b), and crouch (p_c)

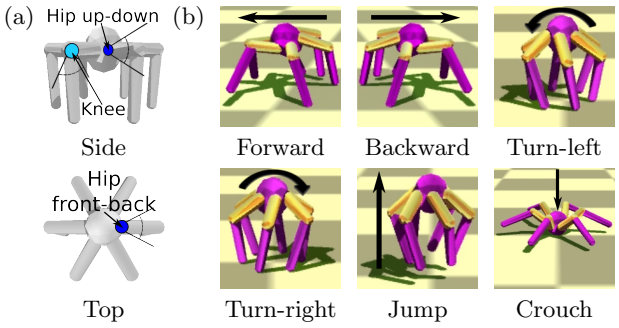


Figure 2: Six-tasks robot and problem. (a) The hexapod robot has 6 knee joints with one degree of freedom and 6 hip joints with 2 degrees of freedom (up-down, front-back). (b) The six-tasks that need to be learned by the robot.

performance are calculated as:

$$p_f = \frac{x_T}{12.5}$$

$$p_b = \frac{-x_T}{12.5}$$

$$p_c = \frac{1}{T}\sum_t (1 - ||\vec{c}_t - [0,0,0]^t||)$$

Let $\vec{x}_t$ define a normal vector attached to the body of the robot, initially aligned with the x-axis, at time t, and let $\vec{y}_t$ be a similar normal vector, initially aligned with the y-axis. Let $\angle_l(\vec{x_1},\vec{x_2}) = \angle(\vec{x_1},\vec{x_2}) \cdot sign(\vec{x_1} \times \vec{x_2})$ be the left angle between $\vec{x_1}$ and $\vec{x_2}$. Let $u_t = 1$ if $\angle(\vec{y}_t,[0,1,0]^t) < \frac{\pi}{3}$, and 0 otherwise, effectively defining whether the robot is upright at time-step t. Let $m(x_1,x_2) = min(x_1,x_2)$. Then the turn-left ($p_l$) and turn-right ($p_r$) performance values can be calculated as:

$$p_l = \frac{25}{T-1}\sum_{t=2}(\angle_l(\vec{x}_t,\vec{x}_{t-1})u_t) + m(1 - \frac{1}{T}\sum_t ||\vec{c}_t - \vec{c}_0||, 0)$$

$$p_r = \frac{25}{T-1}\sum_{t=2}(-\angle_l(\vec{x}_t,\vec{x}_{t-1})u_t) + m(1 - \frac{1}{T}\sum_t ||\vec{c}_t - \vec{c}_0||, 0)$$

These equations can be interpreted as summing the degrees turned by robot while being upright, with a penalty for moving more than 1 unit from the starting location. Lastly, let y_{max} be $max_{t=1}^{T/2}(1 - ||\vec{c}_t - [0,2,0]^t||)$. Jump performance can thus be defined as:

$$p_j = \begin{cases} y_{max} & : y_{max} \cdot u_t \leq 0.5 \\ y_{max} + 1 - ||\vec{p}_T - \vec{c}_0|| & : y_{max} \cdot u_t > 0.5 \end{cases}$$

This equation rewards getting as close as possible to a $[0,2,0]^t$ target coordinate (straight above the starting position) during the first $T/2$ time-steps, and provides a landing bonus based on how close the robot can get to the starting location at the end of the session, provided that the robot is upright and managed to get at least within 0.5 units of the target coordinate. $T = 400$ for p_f and p_b, and $T = 200$ for the other sub-tasks, as they could be evaluated more quickly.

We chose constants such that, when optimizing on a single objective, performance was roughly 1 after 1000 generations. Performance on the problem as a whole is the product of all six sub-tasks, meaning an individual can not ignore any of the sub-tasks completely.

2.3 Evolutionary Algorithms

The pattern recognition experiment is performed with the NSGA-II algorithm [10]. Parameters are the same as in [15], where evolution starts with a randomly generated population of n (here 1000) individuals, with a random individual being a fully connected CPPN without hidden nodes, random weights for its connections (from $[-3,3]$), and random activation functions for its output nodes (sigmoid, Gaussian, linear, or sine). New individuals are created by selecting parents through tournament selection (tournament size 2), and making a copy for each parent. The copies are then mutated, with a 9% chance to add a connection, an 8% chance to remove a connection, a 5% chance to add a node, a 4% chance to remove a node, a 10% per connection chance to change a weight, and a 10% per neuron chance to draw a new activation function.

Every treatment has two or more objectives: a performance objective, a behavioral diversity objective [12, 24], and (optionally) objectives to promote modularity. Behavioral diversity for an individual is calculated by recording a 'behavior vector' from its ANN outputs, binarizing that behavior vector such that values > 0 are 1, and all other values are 0, and then calculating the average Hamming distance to the behavior vector of all other individuals in the population [11]. For the pattern recognition task, the behavior vector is the value of the two ANN outputs over all possible 256 input patterns, producing a binary vector of 512 elements. For the robotics task, we first produce 6 sub-task vectors by recording and binarizing the value of all 18 ANN outputs for the first 5 time-steps of each sub-task, resulting in 6 vectors with 90 elements. We then create a seventh 'majority' vector by taking the element-wise sum of the 6 sub-tasks, and binarizing the result such that values > 3 are 1, and others are 0. To obtain the actual behavior vector, we take the Exclusive-OR of the majority vector with every sub-task vector, and concatenate the 6 resulting vectors. This method encourages different behaviors for each of the sub-tasks, since individuals that ignore their inputs and exhibit the same behavior for each sub-task will all have a behavior vector consisting of zeros.

Depending on the treatment, modularity may be encouraged through a connection cost objective or through direct selection. The connection cost objective is either applied to the ANN, to encourage phenotypic modularity, or the CPPN, to encourage genotypic modularity [7]. The connection cost of an ANN is defined as the summed squared length of all its connections, where the length of a connection is the Euclidean distance between its source and target neurons (possible because every neuron has a geometric coordinate). For CPPN genomes, whose nodes do not have geometric locations, we define the number of connections in the CPPN to be its cost. Because our evolutionary algorithm maximizes objectives, the negative of the cost is added to the list of objectives. As in previous work [7, 15], we simulate the fact that a connection cost should be less important than performance by associating it with a probability p (here 0.75). In any comparison of dominance, there is a p percent chance that the connection cost objective is considered; it is otherwise ignored. For direct selection we calculate the modularity of the relevant network, either the CPPN or the ANN, through a quick approximation [25] of the modularity-Q score for directed networks [18], and we add this score as an objective.

In preliminary work we ran the robotics experiment with NSGA-II, but runs quickly converged to a local optimum, and individuals often focused on a single objective like jumping, while gaining performance on the other sub-tasks by falling over. To avoid these local optima we designed a new algorithm called the Combinatorial Multi-Objective Evolutionary Algorithm (Combinatorial MOEA), which adds every combination of objectives as a separate objective. It is inspired by the work on Innovation Engines, which showed that having a vast number of different objectives improves the ability to perform well on any individual objective, because the path to discover a solution for any challenging objective is unknown ahead of time [26]. For example, perhaps the best path to mastering all six objectives is to first master jumping and moving forward *only*, and then moving on to jumping, moving forward, and crouching, etc.

It is known that Evolutionary Algorithms (and learning animals, including humans) are helped by *staging*, in which a curricula is developed that does not challenge an agent to learn all tasks at once, but creates a path of challenges that build up to the full set (e.g. crawl first, then walk, etc.) [2, 19]. However, it is exceedingly difficult to predict the optimal order or combination of tasks, and computationally expensive to separately try all possible orderings, so our algorithm instead simultaneously rewards all possible combinations of tasks.

With the Combinatorial MOEA, rather than having a single population of n individuals, we instead have s (here 63) sub-populations of n_b (here 100) individuals, referred to as bins. Each of these bins has a set of sub-tasks associated with it, and there exists one bin for every combination of sub-tasks. When an individual is added to one of these bins, its 'performance objective' is set to be the product of its performance on all sub-tasks associated with that bin. A publication on this novel algorithm is in preparation.

To accommodate this design, the following changes were made with respect to NSGA-II. When selecting a parent, we randomly select an individual from the entire population, without tournament selection. We then add a copy of the mutated child to *all* bins. When performing survivor selection, we apply Pareto dominance sorting to every bin separately. Lastly, the number of offspring created per generation is a separate parameter (here 1000), rather than equal to the population size. Both algorithms were implemented in the Sferes 2 framework [23], and all source code is available at http://www.evolvingai.org/ModularAlignment.

2.4 Treatments

We compare HyperNEAT (HN) alone against two treatment categories: those that encourage modularity through selective pressure, and hand-designed treatments whose phenotypes or genotypes have specific decompositions we would expect from modular networks. We have four sources of selective pressure: a connection cost on the phenotype (PCC), a connection cost on the genotype (GCC), direct selection for phenotypic modularity (PMOD), and direct selection for genotypic modularity (GMOD). There are two types of handcrafted networks: an artificially split phenotype (ASP) and an artificially split genotype (ASG). ASP has a phenotypic (ANN) decomposition, such that some neurons are not allowed to connect to other neurons. With ASG, the ANN has a multi-spatial substrate (MSS), such that different parts of the ANN are governed by different CPPN

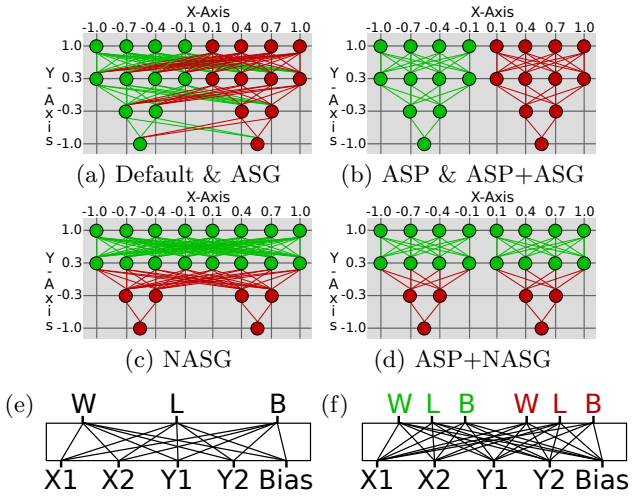
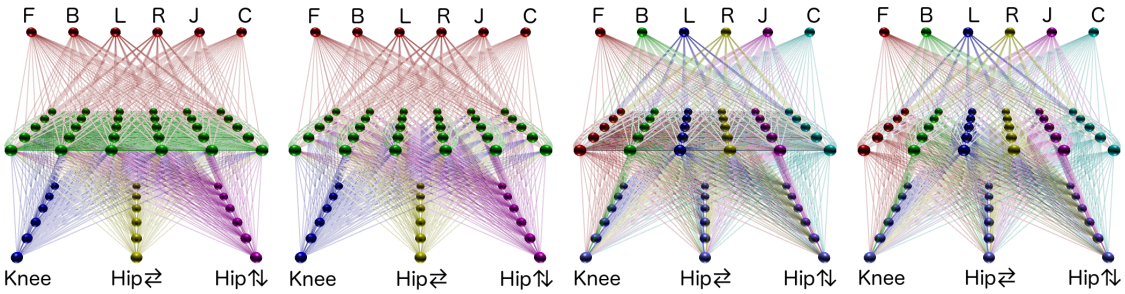


Figure 3: Pattern recognition problem spatial layouts for ANNs and their corresponding CPPNs. Neurons are placed in a 2×2 plane ranging from -1 to 1 in both dimensions. Colors indicate to which space a neuron, connection, or CPPN output belongs in terms of the multi-spatial substrate (MSS), and are thus only meaningful in ASG and NASG treatments. (a) All connections are allowed in the default and ASG setups. The MSS spaces of ASG have a left-right decomposition. (b) ASP does not allow any connections between the left and right side of the problem, which aligns with the ASG decomposition. (c) NASG allows all connections, but features a top-bottom decomposition into MSS spaces. (d) In ASP+NASG, the enforced phenotypic and genotypic decompositions are *not* aligned. (e) The CPPN for this experiment has a weight **W**, LEO **L**, and Bias **B** output. (f) The number of CPPN outputs is doubled in MSS (i.e. ASG and NASG) treatments. The color of the output corresponds to the MSS space they govern.

outputs. In both ASP and ASG, the decompositions are *problem* oriented, such that the decomposition matches the expected structure of the sub-tasks.

In the pattern recognition task, ASP and ASG have a left-right split (Fig. 3a,b). For this task only, we also introduce an ASG variant where the MSS decomposition is *not* aligned with the inherent modularity of the problem (Fig. 3c), which we call the non-aligned artificially split genotype (NASG). If combined with ASP, this results in a genotypic decomposition that is not aligned with the enforced phenotypic split (Fig. 3d). In total, the treatments for the pattern recognition problem are: HN, PCC, GCC, ASP, ASG, ASP+ASG, NASG, and ASP+NASG. PMOD and GMOD treatments were omitted due to computational constraints.

In the robotics task, all treatments have a multi-spatial substrate, meaning that the difference between treatments with and without ASG is based solely on how the ANN is divided into spaces. Unless otherwise noted, the ANN is split in a way that is common for MSS, where input neurons, hidden neurons and different classes of output neurons are all in a separate space (Fig. 4a). The ASP treatments have the same space decomposition, but split the hidden layer into modules by disallowing connections between them such that there exists one module per sub-task (Fig. 4b). In ASG treatments, the MSS spaces, rather than the phenotypic modules, are assigned such that there is one space



(a) Default (b) ASP (c) ASG (d) ASP+ASG

Figure 4: Robotics-problem spatial layouts for ANNs and their corresponding CPPNs. Neurons are placed in a $2 \times 2 \times 2$ cube ranging from -1 to 1 in all dimensions, such that neurons at extreme coordinates are on the edge of this cube. Colors indicate to which space a neuron, connection, or CPPN output belongs in terms of the multi-spatial substrate (MSS). (a) In the default setup, input neurons, hidden neurons, and different classes of actuators belong to different MSS spaces, as is customary for MSS. (b) With ASP, the hidden layer is split into six modules that can not connect. (c) With ASG, MSS spaces are assigned problem wise (left to right). (d) ASP+ASG combines the properties of ASP and ASG.

per sub-task (Fig. 4c). When ASP and ASG are combined, the genotypic decomposition *is* aligned with the phenotypic split (Fig. 4d). In total, the treatments for the robotics problem are: HN, PCC, GCC, PMOD, GMOD, PMOD+GMOD, ASP, ASG, ASP+ASG, ASP+GCC, ASG+PCC. We were unable to combine PCC and GCC in a single treatment because both require a probability $p < 1$, which may result in situations where A dominates B, B dominates C, and C dominates A. Resolving these "dominance triangles" is a topic for future work.

Our results figures show, for each treatment, the median of the best individuals from 30 runs. Shaded areas indicate the 95% bootstrapped confidence interval, and the symbols at the bottom indicate a significant difference ($p < 0.05$, Mann-Whitney U) at that generation versus HyperNEAT.

3. RESULTS AND DISCUSSION

3.1 Pattern Recognition Problem

In the pattern recognition task, HyperNEAT alone shows a steady increase in performance for the first 10000 generations, after which improvements become minor (Fig. 5a). The addition of a connection cost, either applied to the phenotype or the genotype, provides no observable improvement. ASP provides some benefits early on, though the difference with HyperNEAT alone is no longer significant at the final generation. ASG, ASP+ASG and ASP+NASG, on the other hand, perform significantly better than HyperNEAT throughout the run, and achieve perfect performance in the majority of runs. ASG and ASP+ASG both have or evolve aligned genotypic and phenotypic modularity (see below), but ASP+NASG is purposefully misaligned, meaning we can not attribute the success of the ASG treatments to their alignment. We can, however, state that having modular decompositions in both the phenotype and the genotype-phenotype map is beneficial on this problem. In contrast to ASG alone, NASG alone does not perform any better than HyperNEAT, demonstrating that a *problem aligned* genotype is more beneficial than one that is not.

To interpret the results regarding how modular the phenotypes became, it is useful to note that the maximum modularity score for a network with two modules is 0.5. Furthermore, analysis of ANN visualizations (not shown) revealed that all ANNs with a modularity score of 0.5 have the same left-right split as the one enforced by ASP (Fig. 3b).

Even though the left and right problems are completely separate, HyperNEAT alone does not produce a split with maximum modularity, and neither do the PCC or GCC treatments (Fig. 5b). ASG does find a maximally modular split, meaning it naturally evolved phenotypic modularity that is aligned with its genotypic decomposition, and it is the only treatment to do so where this split is not manually enforced. The top-bottom alignment for NASG apparently makes it much harder to find a split with maximum modularity and, as a result, this treatment is significantly less modular than HyperNEAT. Of course, ASP treatments, which have manually enforced modular phenotypic splits, have maximum modularity throughout.

Genotypic modularity rapidly increases for all treatments during the first 1000 generations (Fig. 5c). This increase is almost inevitable considering that the initial, fully-connected ANNs have a modularity of zero, meaning that mutations can only increase (or have no effect on) modularity. The addition of a connection cost is a known method for increasing modularity in networks [7]. However, while GCC increases genomic modularity faster than HyperNEAT alone, this increase stagnates after 1000 generations (Fig. 5c), probably because the connection cost prevents further complexification, since its genomes do not grow as other treatments do (Fig. 5d). This may not be detrimental per se, as it ensures that CPPNs do not grow superfluous structures, but in this instance there was no benefit either (Fig. 5a).

ASP genomes are all significantly more modular than HyperNEAT alone. This difference may emerge because the enforced phenotypic decomposition functions as a template, where genotypic elements that happen to align with one of the phenotypic modules will be preserved, thus slowly developing a modular genotypic structure. This effect is most prominent in treatments that also feature a genotypic split (ASP+ASG and ASP+NASG), possibly because separate pathways are already present in these genotypes; evolution only needs to expand them.

The failure of PCC to perform well and find maximum modularity splits is in contrast to previous results [15], probably because the two sub-problems are no longer mirrored versions of each other. That is, even when the optimal split is found, evolution may be unable to find further increases

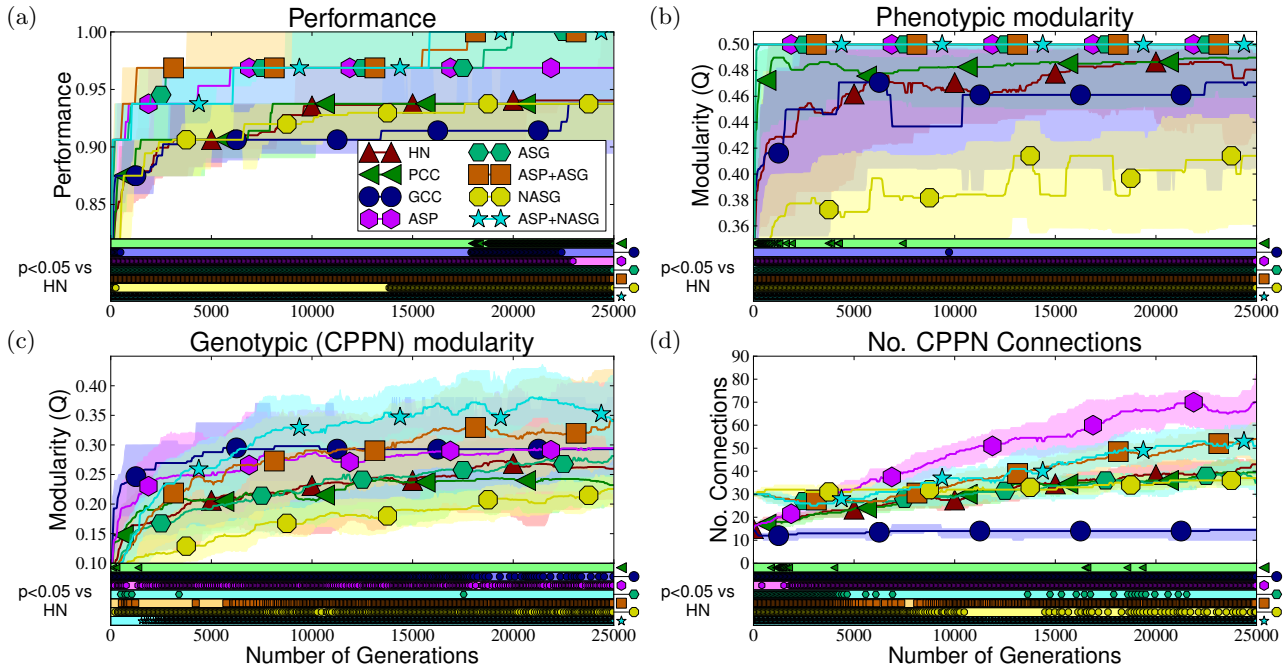


Figure 5: On the pattern recognition task, treatments with ASG perform significantly better, and have a significantly higher phenotypic modularity, than HyperNEAT alone. (a) The ASG, ASP+ASG, and ASP+NASG, treatments all perform significantly better than HyperNEAT alone after 5000 generations. (b) ASG is the only non-ASP treatment that achieves maximum modularity in a majority of runs. (c) Despite the addition of a connection cost, GCC quickly stagnates in genotypic modularity. (d) In contrast to other treatments, GCC genomes do not increase in size.

in performance without introducing irregularities that break the left-right split. This is an important hint that encouraging phenotypic modularity alone is not sufficient to solve tasks that require several different modules.

3.2 Robotics Problem

While PMOD and GMOD alone perform poorly on the robotics problem, PMOD+GMOD performs significantly better than HyperNEAT alone (Fig. 6a). This result strongly hints at the importance of encouraging both genotypic and phenotypic modularity. Neither PCC nor GCC result in a significant performance difference when compared to HyperNEAT, possibly because they do not encourage phenotypic and genotypic modularity simultaneously. The observed fitness values may seem low, but visual inspection of robot behaviors indicates that sub-tasks are being solved; an individual with a fitness of 0.008 qualitatively solves all tasks.

The PMOD and GMOD treatments have very high phenotypic and genotypic modularity, respectively (Fig. 6b, c), but their low performance (Fig. 6a) suggests that this modularity is non-functional; they are modular for the sake of being modular, but they do not gain the benefits usually associated with modularity. When these objectives are combined (PMOD+GMOD), the modularity in both the genotype and the phenotype does decrease, but it remains significantly higher than HyperNEAT alone. PCC produces phenotypes that are also significantly more modular than HyperNEAT, but this improvement is very small when compared to the impact of PMOD+GMOD (Fig. 6b). In contrast to the pattern recognition problem, genotypic modularity hardly increases for any but the treatments that explicitly select for it (Fig. 6c), probably because the genomes are initially much larger and evolution mostly tweaks CPPN weights rather than the CPPN architecture, as is suggested by the lack of change in CPPN size (Fig. 6d). In this case, GCC is not the only treatment that lacks meaningful complexification.

The increased modularity of the PMOD+GMOD treatment is also visually apparent, with some networks dividing into clearly separated communities (Fig. 6e, f). These communities are quite different from the ones manually imposed upon the network in the ASP treatment, and their function is not immediately clear from observation. Videos (available at http://www.evolvingai.org/ModularAlignment) reveal that modules often separate a pattern-generator section of the network from neurons with a static behavior. Because testing for alignment is non-trivial, it is unknown whether the observed phenotypic modularity is aligned with the underlying genotypic modularity, but this analysis will be included in future work.

Our hand-designed treatments perform no different from HyperNEAT alone, suggesting that we have not managed to capture the advantages of a modular phenotype or genotype in these treatments (Fig. 7a). It is noteworthy that ASG treatments have a significantly higher *phenotypic* modularity than HyperNEAT, showing that structure in the genotype alone can affect phenotypic organization, though the modularity is still much lower than what is achieved by PMOD+GMOD (Fig. 7b). Also, genotypic modularity is extremely low for all hand-designed treatments (Fig. 7c).

There may be several reasons for why our hand-designed networks perform so poorly. First, it is possible that 5000 generations is simply not enough to reveal significant differences. In preliminary experiments, we extended the number of generations to 15000 for the HN, ASG and ASG+ASP treatments, but no significant differences arose. Alternatively, it is possible that we chose poor manual splits, but

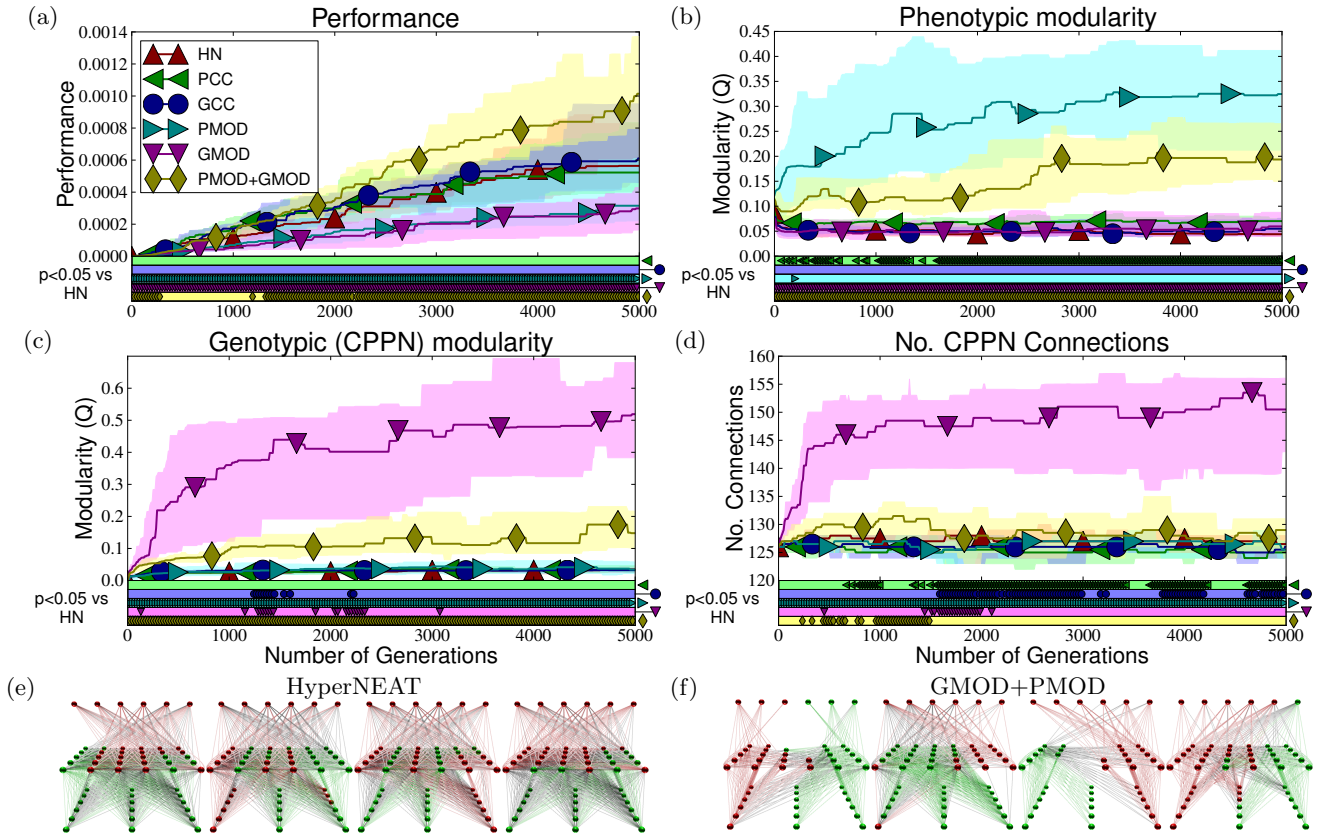


Figure 6: On the robotics task, direct selection for both phenotypic and genotypic modularity produces genotypes and phenotypes that are significantly more modular (b, c), and that perform significantly better (a) than HyperNEAT. (a) PMOD and GMOD alone perform significantly worse than HyperNEAT. (b, c) While PMOD+GMOD achieves lower phenotypic and genotypic modularity than PMOD and GMOD, respectively, its modularity is still significantly higher than HyperNEAT's in both cases. (d) Genotypes do not appear to complexify on the robotics problem, with the exception of GMOD's, probably because larger networks can attain a higher modularity score. (e, f) The four best performing networks for the HyperNEAT and GMOD+PMOD treatments, where nodes are colored according to the modules discovered by Leicht and Newman's graph partitioning algorithm for directed networks [18]. This method detects groups of nodes that are strongly connected to each other, while being only sparsely connected to nodes in other groups. HyperNEAT ANNs look fully connected, but some GMOD+PMOD ANNs contain clear modular decompositions.

that there exist other modular configurations that would be more beneficial. This hypothesis is supported by the fact that the PMOD+GMOD treatment does lead to increased modularity and performance, yet provides decompositions that are quite different from the one we imposed (Fig. 6e, f). This demonstrates the importance of techniques that can find modular decompositions automatically, since manual designs may not be optimal.

4. CONCLUSION

Our results suggest that, when different functional modules are required within the same neural network, which is the case for all challenging problems, performance benefits can be gained by simultaneously encouraging modularity in both the genotype and the phenotype. They further indicate that the role of genotypic and phenotypic *alignment* is still unclear. In addition, we have shown that, while a task-oriented decomposition worked well for a simple pattern recognition problem, such a decomposition was not beneficial on a more challenging robotics problem, emphasizing the importance of automatic, rather than manual, methods for generating modular architectures. Lastly, on the robotics task, even the best performing treatment remains orders of magnitude below what is achievable by optimizing the sub-tasks separately, indicating that the problem is far from solved. Overall, our work highlights that much more research is required before we can successfully evolve structurally organized neural networks that solve tasks that require multiple, different neural modules.

Acknowledgements: We would like to thank Kai Olav Ellefsen, Henok Mengistu, and all members of the Evolving AI Lab for their feedback and support. Support came from an NSF CAREER award (CAREER: 1453549) to JC, the ANR Creadapt award (ANR-12-JS03-0009) to JBM, and via computers provided by the US NSF and DOE's Open Science Grid.

5. REFERENCES

[1] L. Altenberg. Modularity in evolution: some low-level questions. *Modularity: understanding the development and evolution of complex natural systems*, pages 99–128, 2005.

[2] J. Auerbach and J. C. Bongard. How robot morphology and training order affect the learning of multiple behaviors. *Proc. IEEE Cong. Evolutionary Comput*, pages 39–46, May 2009.

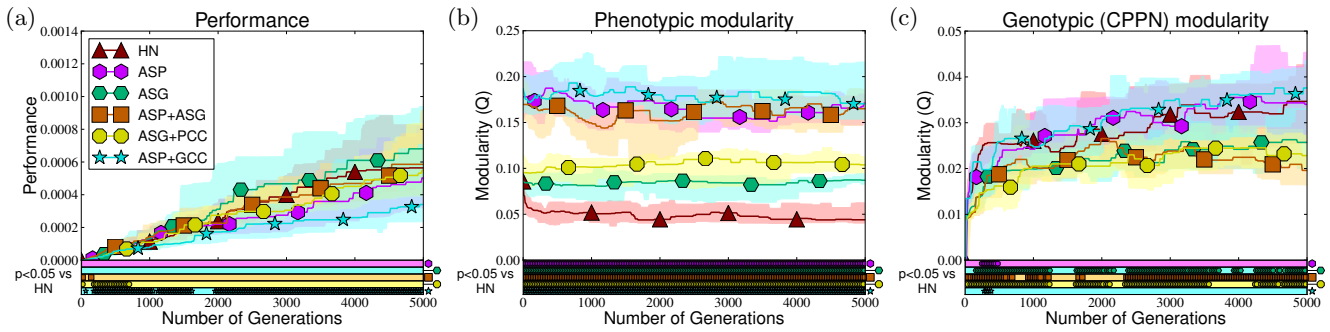


Figure 7: (a) On the robotics task, artificially splitting the genotype or phenotype does not lead to significant increases in performance as compared to HyperNEAT. (b) ASG treatments produce phenotypes that are significantly more modular than HyperNEAT alone, though the difference is small. (c) ASG treatments produce genotypes that are significantly less modular than HyperNEAT alone, possibly because the genotype is already split.

[3] R.D. Beer and J.C. Gallagher. Evolving dynamical neural networks for adaptive behavior. *Adaptive behavior*, 1(1):91, 1992.

[4] E. Bullmore and O. Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature Reviews Neuroscience*, 10(3):186–198, 2009.

[5] S.B. Carroll. Chance and necessity: the evolution of morphological complexity and diversity. *Nature*, 409(6823):1102–1109, 2001.

[6] J. Clune, B.E. Beckmann, C. Ofria, and R.T. Pennock. Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In *Proc. IEEE Cong. Evolutionary Comput*, pages 2764–2771, 2009.

[7] J. Clune, J-B. Mouret, and H. Lipson. The evolutionary origins of modularity. *Proc. Royal Society B*, 280(20122863), 2013.

[8] J. Clune, K.O. Stanley, R.T. Pennock, and C. Ofria. On the performance of indirect encoding across the continuum of regularity. *IEEE Trans. Evolutionary Comput*, 15(4):346–367, 2011.

[9] E.H. Davidson and D.H. Erwin. Gene regulatory networks and the evolution of animal body plans. *Science*, 311(5762):796, 2006.

[10] K. Deb, A. Pratap, S. Agarwal, and T.A.M.T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Trans. Evolutionary Comput*, 6(2):182–197, 2002.

[11] S. Doncieux and J.-B. Mouret. Behavioral diversity measures for evolutionary robotics. In *Proc. IEEE Cong. Evolutionary Comput*, pages 1–8. IEEE, 2010.

[12] S. Doncieux and J.-B. Mouret. Beyond black-box optimization: a review of selective pressures for evolutionary robotics. *Evolutionary Intelligence*, 7(2):71–93, 2014.

[13] K. O. Ellefsen, J.-B. Mouret, and J. Clune. Neural modularity helps organisms evolve to learn new skills without forgetting old skills. *PLoS Comput Biol*, 11(4):e1004128, 2015.

[14] G.S. Hornby and J. B. Pollack. Evolving L-systems to generate virtual creatures. *Computers & Graphics*, 25(6):1041–1048, December 2001.

[15] J. Huizinga, J. Clune, and J.-B. Mouret. Evolving neural networks that are both modular and regular: Hyperneat plus the connection cost technique. In *Proc. Genetic & Evolutionary Comput. Conf*, pages 697–704. ACM, 2014.

[16] N. Kashtan and U. Alon. Spontaneous evolution of modularity and network motifs. *Proc. Nat'l Acad. Sciences*, 102(39):13773–13778, September 2005.

[17] C.P. Klingenberg. Developmental constraints, modules and evolvability. *Variation: A central concept in biology*, pages 1–30, 2005.

[18] E. A. Leicht and M. E. J. Newman. Community structure in directed networks. *Physical review letters*, 100(11):118703, 2008.

[19] M. A. Lewis, A. H. Fagg, and A. Solidum. Genetic programming approach to the construction of a neural network for control of a walking robot. In *Proc. IEEE Int. Conf. Robotics Automation*, pages 2618–2623. IEEE, 1992.

[20] H. Lipson. Principles of modularity, regularity, and hierarchy for scalable systems. *Journal of Biological Physics and Chemistry*, 7(December):125–128, 2007.

[21] H. Mengistu, J. Huizinga, J.-B. Mouret, and J. Clune. The evolutionary origins of hierarchy. *arXiv preprint arXiv:1505.06353*, 2015.

[22] J.-B. Mouret and S. Doncieux. Evolving modular neural-networks through exaptation. In *Proc. IEEE Cong. Evolutionary Comput*, pages 1570–1577. IEEE, 2009.

[23] J-B. Mouret and S. Doncieux. Sferes v2: Evolvin' in the multi-core world. *IEEE Cong. Evolutionary Comput*, (2):1–8, July 2010.

[24] J.-B. Mouret and S. Doncieux. Encouraging behavioral diversity in evolutionary robotics: an empirical study. *Evolutionary Computation*, 1(20), 2012.

[25] M.E.J. Newman. Modularity and community structure in networks. *Proc. Nat'l Acad. Sciences*, 103(23):8577–8582, 2006.

[26] A. M. Nguyen, J. Yosinski, and J. Clune. Innovation engines: Automated creativity and improved stochastic optimization via deep learning. In *Proc. Genetic & Evolutionary Comput. Conf*, pages 959–966. ACM, 2015.

[27] M. E. Palmer. Evolved neurogenesis and synaptogenesis for robotic control: the l-brain model. In *Proc. Genetic & Evolutionary Comput. Conf*, pages 1515–1522. ACM, 2011.

[28] J. K. Pugh and K. O. Stanley. Evolving multimodal controllers with hyperneat. In *Proc. Genetic & Evolutionary Comput. Conf*, pages 735–742. ACM, 2013.

[29] O. Sporns and R. F. Betzel. Modular brain networks. *Annual review of psychology*, 67(1), 2015.

[30] K.O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162, 2007.

[31] K.O. Stanley, D.B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009.

[32] G.F. Striedter. *Principles of brain evolution*. Sinauer Associates Sunderland, MA, 2005.

[33] P. Verbancsics and K.O. Stanley. Constraining connectivity to encourage modularity in hyperneat. In *Proc. Genetic & Evolutionary Comput. Conf*, pages 1483–1490. ACM, 2011.