Constraining Connectivity to Encourage Modularity in HyperNEAT

In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011).* New York, NY: ACM, 2011

Phillip Verbancsics
Department of EECS
University of Central Florida
Orlando, FL 32816, USA
verb@eecs.ucf.edu

Kenneth O. Stanley
Department of EECS
University of Central Florida
Orlando, FL 32816, USA
kstanley@eecs.ucf.edu

ABSTRACT

A challenging goal of generative and developmental systems (GDS) is to effectively evolve neural networks as complex and capable as those found in nature. Two key properties of neural structures in nature are regularity and modularity. While HyperNEAT has proven capable of generating neural network connectivity patterns with regularities, its ability to evolve *modularity* remains in question. This paper investigates how altering the traditional approach to determining whether connections are expressed in HyperNEAT influences modularity. In particular, an extension is introduced called a Link Expression Output (HyperNEAT-LEO) that allows HyperNEAT to evolve the pattern of weights *independently from* the pattern of connection expression. Because HyperNEAT evolves such patterns as functions of geometry, important general topographic principles for organizing connectivity can be seeded into the initial population. For example, a key topographic concept in nature that encourages modularity is *locality*, that is, components of a module are located near each other. As experiments in this paper show, by seeding HyperNEAT with a bias towards local connectivity implemented through the LEO, modular structures arise naturally. Thus this paper provides an important clue to how an indirect encoding of network structure can be encouraged to evolve modularity.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Performance, Experimentation

Keywords

Generative and Developmental Systems, Artificial Neural Networks, Modularity, HyperNEAT

1. INTRODUCTION

As generative and developmental systems (GDS) are asked to evolve increasingly large and complex structures, the question of how to organize and constrain such structures becomes increasingly important. Neural networks in nature exhibit several important organizational principles, such as modularity, regularity, and hierarchy [11, 13, 22], which enable evolution to scale structures in complexity and size. Such features might accordingly enable evolutionary algorithms similarly to scale to problems of high complexity and dimensionality if they could be encouraged to emerge. This paper shows how modularity can be encouraged as a *function of geometry* through an indirect encoding.

The challenge in encouraging modularity is to bias search towards modular solutions without restricting the search space to modularity. The danger in explicitly designing modularity into the representation is that it may bias search towards modular structure when non-modular designs may be required, inhibiting search from finding an optimal solution [7]. For example, fully-connected artificial neural networks (ANNs) are completely non-modular. To produce a modular ANN, connectivity would be restricted, but that would prevent solutions that may require full connectivity.

Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT; [9, 10, 15]) has been shown to generate regularities in ANN weight patterns [3, 9, 10, 15, 21], but Clune et al. [1] showed that it struggles to produce modular patterns. To address this problem, this paper introduces a novel insight that can consistently bias HyperNEAT towards modular structures. The initial idea is that a bias towards *locality* is an important prerequisite to modularity. However, locality alone is not enough because not *all* connectivity should be local. Thus the key insight is that the search should *start* with a bias towards locality that can later be adjusted by evolution to different levels for different parts of the network geometry. This idea is implemented and tested in this paper in a HyperNEAT variant called HyperNEAT with Link Expression Output (HyperNEAT-LEO).

To demonstrate its advantage, HyperNEAT-LEO is compared to the standard HyperNEAT method (with and without locality bias) and another variant that enforces an external distribution on connectivity. The HyperNEAT variants are investigated in the Retina Left and Right problem [1] (based originally on Kashtan and Alon [12], in which the ANN must indicate whether particular objects are detected in the left and right retinas). The Retina problem is modular because the calculations for the left and right sides are independent, which has proven difficult for HyperNEAT to represent [1]. In this task, a significant difference is demonstrated among the variants. The standard HyperNEAT does not perform well at the task, finding a solution only 12% of the time or less. In addition, both biasing standard Hy-

GECCO'11, July 12–16, 2011, Dublin, Ireland.

perNEAT towards locality in *weights* and imposing a local connectivity distribution perform even worse and never find a solution. However, interestingly, HyperNEAT-LEO, which evolves the pattern of expressed connections, is able to achieve up to a 91% success rate if it begins with a bias towards local connectivity. These results demonstrate that modularity can be encoded as a function of geometry, providing a means to evolving modularity in HyperNEAT.

2. BACKGROUND

The geometry-based methods that underlie the LEO concept are first reviewed and then modularity is discussed.

2.1 NeuroEvolution of Augmenting Topologies (NEAT)

This section briefly reviews the NEAT evolutionary algorithm [16, 18], a popular policy search method that evolves ANNs. It is also the foundation for the HyperNEAT method. NEAT evolves connection weights as well as adds new nodes and connections over generations, thereby increasing solution complexity. It has been proven to be effective in challenging control and decision-making tasks [18, 19, 23, 24]. NEAT starts with a population of small, simple ANNs that increase their complexity over generations by adding new nodes and connections through mutation. That way, the topology of the network does not need to be known a priori; NEAT searches through increasingly complex networks as it evolves their connection weights to find a suitable level of complexity. The techniques that facilitate evolving a population of diverse and increasingly complex networks are described in detail in Stanley and Miikkulainen [16] and Stanley and Miikkulainen [18]; the important concept for the approach in this paper is that NEAT is a policy search method that discovers the right topology and weights of a network to maximize performance on a task. The next section reviews the extension of NEAT called HyperNEAT that allows it to exploit geometry through representation.

2.2 CPPNs and HyperNEAT

NEAT is an appropriate platform to study modularity in GDS because it easily extends to an *indirect encoding*, which means a *compressed* description of the solution network. Such compression makes the policy search practical even if the state space is high-dimensional. One effective indirect encoding is to compute the network structure as a function of geometry. This section describes such an extension of NEAT, called Hypercube-based NEAT (HyperNEAT [9, 10, 15]), which enables high-dimensional indirect representations. The effectiveness of the geometry-based learning in HyperNEAT has been demonstrated in multiple domains and representations, such as checkers [9, 10], multi-agent predator prey [4, 6], quadruped locomotion [2], and RoboCup Keepaway [21]. A full description of HyperNEAT is in Gauci and Stanley [9, 10] and Stanley et al. [15].

The main idea in HyperNEAT is that geometric relationships are learned though an indirect encoding that describes how the *weights* of the ANN can be *generated* as a function of geometry. Unlike a *direct* representation, wherein every connection in the ANN is described individually, an indirect representation describes a pattern of parameters without explicitly enumerating each such parameter. That is, information is reused in such an encoding, which is a major focus in the field of GDS from which HyperNEAT originates [17, 20]. Such information reuse allows indirect encoding to search a compressed space. HyperNEAT discovers the *regularities* in the geometry and learns a policy based on them.

The indirect encoding in HyperNEAT is called a *compositional pattern producing network* (CPPN; [14]), which encodes the *weight pattern* of an ANN [9, 15]. The idea behind CPPNs is that geometric patterns can be encoded by a *composition of functions* that are chosen to represent common regularities. Given a function f and a function g, a composition is defined as $f \circ g(x) = f(g(x)))$. In this way, a set of simple functions can be composed into more elaborate functions through hierarchical composition (e.g. $f \circ g(f(x) + g(x)))$. For example, the Gaussian function is symmetric, so when it is composed with any other function, the result is a symmetric pattern. The internal structure of a CPPN is a weighted network, similar to an ANN, that denotes which functions are composed and in what order. The appeal of this encoding is that it can represent a pattern of connectivity, with regularities such as symmetry, repetition, and repetition with variation, through a network of simple functions (i.e. the CPPN), which means that instead of evolving ANNs, NEAT can evolve *CPPNs* that generate ANN weight patterns.

Formally, CPPNs are *functions* of geometry (i.e. locations in space) that output connectivity patterns whose nodes are situated in n dimensions, where n is the number of dimensions in a Cartesian space. Consider a CPPN that takes four inputs labeled x_1, y_1, x_2, and y_2; this point in four-dimensional space can *also* denote the connection between the two-dimensional points (x_1, y_1) and (x_2, y_2). The output of the CPPN for that input thereby represents the weight of that connection (figure 1). By querying every pair of points in the space, the CPPN can produce an ANN, wherein each queried point is the position of a neuron. While CPPNs are themselves networks, the distinction in terminology between CPPN and ANN is important for explicative purposes because in HyperNEAT, CPPNs *encode* ANNs. Because the connection weights are produced as a function of their endpoints, the final pattern is produced with *knowledge* of the domain geometry, which is literally depicted geometrically within the constellation of nodes.

In effect, the CPPN paints a pattern within a n-dimensional hypercube that is interpreted as an isomorphic connectivity pattern. Weight patterns produced by a CPPN in this way are called *substrates* so that they can be verbally distinguished from the CPPN itself. It is important to note that the structure of the substrate is independent of the structure of the CPPN. The substrate is an ANN whose nodes are situated in a coordinate system, while the CPPN defines the connectivity among the nodes of the ANN. The experimenter defines both the location and role (i.e. hidden, input, or output) of each node in the substrate.

As a rule of thumb, nodes are placed on the substrate to reflect the geometry of the domain (i.e. the state), making setup straightforward [2, 10, 15]. For example, a visual field can be laid out in two dimensions such that nodes that receive input from adjacent locations in the image are literally adjacent in the network geometry. This way, the pattern of weights becomes a direct function of the domain geometry, which means that knowledge about the problem can be injected into the search and HyperNEAT can exploit the regularities (e.g. adjacency, or symmetry, which the CPPN sees) of a problem that are invisible to traditional encod-

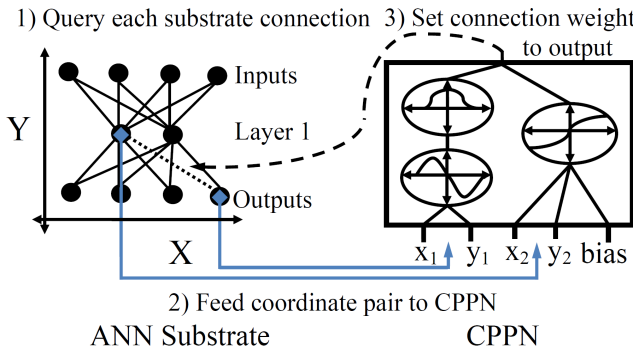

Figure 1: A CPPN Describes Connectivity. A grid of nodes, called the ANN *substrate*, is assigned coordinates. (1) Every connection between layers in the substrate is queried by the CPPN to determine its weight; the dotted line connecting layers in the substrate represents a sample such connection. (2) For each such query, the CPPN inputs the coordinates of the two endpoints, which are highlighted in the substrate. (3) The weight between them is output by the CPPN. Thus, CPPNs can generate regular patterns of connections.

Input: Substrate Configuration
Output: Solution CPPN
Initialize population of minimal CPPNs with random weights;
while *Stopping criteria is not met* **do**
  foreach *CPPN in the population* **do**
    foreach *Possible connection in the substrate* **do**
      Query the CPPN for weight w of connection;
      if *Abs(w) > Threshold* **then**
        Create connection with a weight scaled proportionally to w (figure 1);
      end
    end
    Run the substrate as an ANN in the task domain to ascertain fitness;
  end
  Reproduce CPPNs according to the NEAT method to produce the next generation;
end
Output the champion CPPN.

Algorithm 1: Basic HyperNEAT Algorithm

ings. For example, geometric knowledge can be imparted e.g. by including a hidden node in the CPPN that computes $Gauss(x_2 - x_1)$, which encodes the concept of locality and symmetry on the x-axis, an idea employed in the implementation in this paper. This idea can be illustrated by considering three source points (i.e. nodes) with x positions $-1, 0, 1$ and one target point at $x = 0$:

$$w_1 = Gauss(0 + 1) = 0.37,$$
$$w_2 = Gauss(0 - 0) = 1,$$
$$w_3 = Gauss(0 - 1) = 0.37.$$

Thus, the output of this function is both symmetric about the x-axis and decreasing with distance from the target.

In summary, HyperNEAT evolves the internal topology and weights of the CPPN that compactly *encodes* ANN weight patterns. The complete process is shown in Algorithm 1. Next, modularity is discussed.

2.3 Modularity

Modularity can be defined as the separability of structure into functionally independent units [1, 12, 13]. Modularity is common to both biological and engineering systems. For example, cells assemble into tissues, tissues into organs, and organs into organisms. Decomposing a design into modules can allow evolutionary algorithms to address complex tasks that require sub-functions of the solution to be independently optimized [12]. Approaches to modularity in evolutionary algorithms range from encapsulating modules [7, 25] to alternating tasks to facilitate their discovery [12].

A common approach to modularity in evolutionary algorithms is to preserve or create modules through high-level processes. For example, *modularity-preserving representations* preserve the modularity that exists in the phenotype space by transferring it to the genotype space through encapsulation [7, 25]. The process of encapsulation creates new genotypic primitives that are groupings of existing primitives. In this way, modules can be generated and assembled together to produce solutions. Such high-level processes for modularity are dependent on the capability of the algorithm to select beneficial modules and may be detrimental to search by creating modules that are not useful.

Alternatively, the evolutionary algorithm may be modified such that fitness pressure is applied to encourage discovering modularity. In *modularly varying goals* evolution [12], the task that is being solved alternates at set intervals during evolution. Bias towards modularity is created by selecting tasks in which there are shared sub-goals and it is easier to switch between tasks if the appropriate modules for these sub-goals are discovered. For example, the modularly varying goals version of the Retina Problem [12] switches between the tasks of (1) determining if there is a target pattern in the right *or* left retina and (2) determining if there is a target pattern in the right *and* left retina. These tasks must independently discover the appropriate functions for the left and right retinas, and then combine them correctly depending on the task. The challenge is selecting tasks that share modules such that their discovery is beneficial.

Clune et al. [1] applied HyperNEAT to the modularly-varying goals Retina Problem. Standard HyperNEAT struggled to produce modularity in this task and achieved significantly lower correct classifications than the direct encoding implemented by Kashtan and Alon [12]. In addition to the modularly-varying goals version of the Retina Problem, HyperNEAT was also tested in the non-varying Retina Problems, i.e. the fixed goal of right *and* left and the fixed goal of right *or* left, and alternative Retina Problems that must judge each retina's input individually. For example, a left output indicates validity for the left pattern. The most difficult of these tasks for HyperNEAT is the Retina Left and Right task [1], which must correctly judge as valid or invalid each retina's pattern separately. Interestingly, Clune et al. [1] found that *imposing* modularity by disabling connections between the left and right sides of the network improved HyperNEAT's performance [1]. This enforced modularity demonstrates that modularity could potentially provide HyperNEAT an advantage in this task, but that HyperNEAT struggles to create modularity on its own.

This paper adds to our understanding of modularity by focusing on the role of geometry in encoding modularity. The next section explains modifications to HyperNEAT based on this idea that alter how connection expression is controlled.

3. THRESHOLDING IN HYPERNEAT

Although the HyperNEAT method succeeds in a number of challenging tasks [9, 10, 15, 21] by exploiting geometric regularities in the solution, it is less effective at generating modular solutions [1]. Instead it tends to generate fully-connected networks even when limited connectivity is beneficial. The conventional method for controlling connectivity in HyperNEAT is a threshold that limits the range of values output by the CPPN that can be expressed as weights [10, 15]. The threshold is a parameter specified at initialization that is uniformly applied to all connections queried. When the magnitude of the output of the CPPN is below this threshold, the connection is not expressed. In this way, the proportion of connections expressed may be controlled by raising or lowering the threshold when fewer or more connections are desired. Because the threshold is *uniform* across all connections, it influences every connection equally with no bias towards modularity (which likely requires different levels of connectivity in different areas).

As an alternative to the traditional uniform threshold, one idea to control connection expression is a *dynamic threshold*. The dynamic threshold varies depending on the specific nodes or the connection being queried. As each connection is queried, the threshold, which determines whether the CPPN-generated weight should be expressed as a connection, is updated and then applied. In this way, a particular expression distribution that is determined a priori can be enforced to influence the pattern of connectivity in the ANN. In particular, this paper explores a dynamic threshold based on distance with the value $t = c + c * ||\mathbf{n}_i - \mathbf{n}_j||$, where t is the local threshold value, c is a constant parameter set at initialization and $||\mathbf{n}_i - \mathbf{n}_j||$ is the Euclidean norm of the difference between the endpoint-nodes' coordinate-vectors. This threshold makes sense for modularity because it expresses the concept of *locality* by discouraging connections of greater distance. That is, the higher the distance is between nodes, the higher the threshold will be.

In another alternative to the uniform threshold, instead of thresholding connections through an external value, HyperNEAT itself is extended to generate an *expression pattern* that controls whether connections are expressed at different locations independently of their weights. For this purpose, a new Link Expression Output (LEO) is added to the CPPN that indicates whether a connection is expressed. When the LEO (which is a step function) is greater than 0.0 the connection weight is set to the usual CPPN weight output; otherwise it is not expressed (Algorithm 2). In this way, the pattern of connectivity can be controlled by the CPPN itself *independently of the weights*, and that connectivity can be evolved as a function of geometry. This approach follows the idea that connection expression and weights are best considered separately.

Furthermore, because expression patterns are functions of geometry (as are the weights), evolution can be seeded with geometric principles that bias the pattern of connectivity. Evolution is seeded by initializing CPPNs in the first generation with seed topographies that represent important concepts. For example, locality can be expressed through

```
Input: Substrate Configuration
Output: Solution CPPN
Initialize population of minimal CPPNs with random weights;
while Stopping criteria is not met do
    foreach CPPN in the population do
        foreach Possible connection in the substrate do
            Query the CPPN for weight w of connection and expression value e;
            if e > 0.0 then
                Create connection with a weight w;
            end
        end
        Run the substrate as an ANN in the task domain to ascertain fitness;
    end
    Reproduce CPPNs according to the NEAT method to produce the next generation;
end
Output the champion CPPN.
```

Algorithm 2: HyperNEAT-LEO Algorithm

a Gaussian function. Because the Gaussian function peaks when its input is 0.0, inputting a difference between coordinates, e.g. Δx, achieves the highest value when the coordinates are the same. To seed for locality on multiple dimensions, a Gaussian node can be added to the seed for each such dimension, as shown in figure 2. In this way, such seeds provides the concept of locality because the more local the connection (i.e. Δx close to 0.0), the greater the impact of the Gaussian function. This extension is called HyperNEAT with a Link Expression Output (HyperNEAT-LEO).

Each of the three variants discussed in this section provides different means of controlling connectivity and encouraging modularity. The next section introduces the experiments designed to demonstrate the differences among these approaches to thresholding.

4. EXPERIMENTAL SETUP

The experiments in this paper are designed to investigate the effect of different methods of thresholding on connectivity and modularity. The key focus is on the idea that an effective means of expressing modularity is as a *function of geometry*. This geometric function may be externally defined or evolved as a geometric pattern expressed by the CPPN. One key to modularity is limiting the number of connections, but limiting the connections alone does not necessarily lead to modularity. This section explains the task designed to explore how potential thresholding approaches influence modularity.

Introduced by Kashtan and Alon [12], the Retina Problem poses the challenge of creating an ANN that must identify patterns input into the left and right retinas. The retinas consist of four inputs each that are each set to one of two values. Because the values for each input are binary, a four-input retina can take 16 possible patterns and there are overall 256 configurations for two four-input retinas. The individual retinas each have a unique set of specified patterns identified as valid or invalid that are equally divided into eight patterns each.

In the original Retina Problem introduced by Kashtan and Alon [12], the ANN is responsible for determining ei-

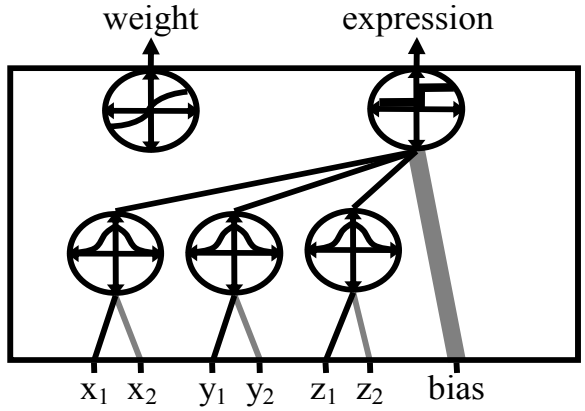

Figure 2: Seed CPPN Expressing Global Locality. Initial CPPN topologies imparted with geometric principles can seed evolution. For example, a six-dimensional CPPN can be initialized with three hidden nodes connected by positive (black) and negative (gray) weights. These hidden nodes take as input $x_1 - x_2$ (Δx), $y_1 - y_2$ (Δy), and $z_1 - z_2$ (Δz) respectively and have Gaussian activation functions. The nodes are connected to the LEO, which has a bias of -3. Because the Gaussian function peaks at 0.0, the expression output value is greater than or equal to 0.0 only when Δx, Δy, and Δz are 0.0. Thus, the initial population is seeded with CPPNs that constrain connectivity to respect locality. When seeding in this way, the weight output of the CPPN is included as usual with a set of direct connections from all the inputs (not shown).

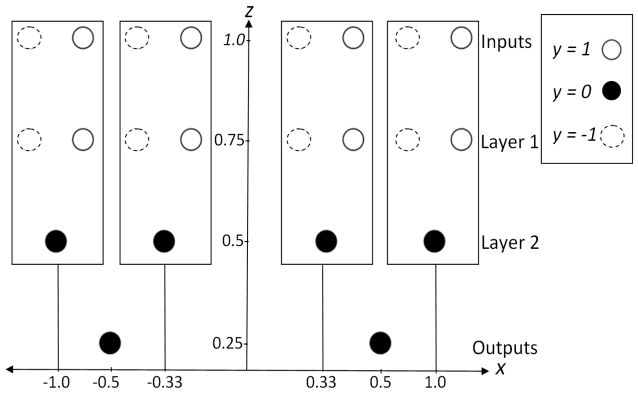

Figure 3: Substrate Configuration for the Retina Problem. The substrate configuration is identical to the setup of Clune et al. [1]. The substrate consists of four layers, with the inputs at $z = 1.0$, the first hidden layer at $z = 0.75$, the second hidden layer at $z = 0.5$, and the outputs at $z = 0.25$. Feed-forward connections are allowed between neighboring layers. Nodes in each z layer are placed at an x, y-coordinate. The y coordinates are indicated by the different circle patterns. These patterns are solid-line circles for $y = 1.0$, filled black circles for $y = 0.0$ and dotted line circles for $y = -1.0$. The left and right sides of the retina are reflected through symmetric coordinates on the negative and positive sides the x-axis. This configuration provides a definite division between left and right modules at $x = 0$.

ther (1) the left *and* right patterns are valid, or (2) the left *or* right patterns are valid. In their version, modularity is helpful when alternating between these tasks because the functions of correctly classifying the left and right patterns are combined. An extension of these tasks, the Retina Left and Right task introduced by Clune et al. [1], removes the need to combine the decision of the modules with a logical operation; instead the ANN must separately judge as valid or invalid each retina's pattern. Standard HyperNEAT was applied to this task and was shown to struggle with producing the modularity required [1]. In fact, the most difficult of the variant tasks of Kashtan and Alon [12] for HyperNEAT in Clune et al. [1] was the Retina Left and Right task, which is thus chosen to explore modularity in this paper.

As shown in figure 3, the substrate is configured with four layers consisting of eight input nodes, eight hidden nodes in layer one, four hidden nodes in layer two, and two output nodes. Their geometric coordinates follow Clune et al. [1]. Note that this substrate has three dimensions (x, y, z). Each of the inputs are set to either -3.0 or 3.0, indicating off or on for each retina input. The left and right outputs specify the classification for the left and right retinas, respectively, where values close to -1.0 indicate invalid and values close to 1.0 indicate valid inputs. Each of the 256 possible patterns is presented to the retinas and the fitness of the solution is inversely proportional to the summed distance of the outputs from the correct values for all patterns. This setup follows Clune et al. [1].

Eight treatments for controlling thresholding were evaluated on this task. The **uniform threshold** is evaluated with both low (0.3) and high (1.3) threshold settings. The **dynamic distance threshold** is evaluated with a constant of 0.45. Finally, three treatments for **HyperNEAT-LEO** are examined in which evolution is not seeded, evolution is seeded with a global locality pattern, and evolution is seeded with a locality pattern only along the x-axis. The seeded locality pattern is a CPPN with three hidden nodes with Gaussian activation functions that input Δx, Δy, and Δz individually. These hidden nodes are then connected to the output that specifies whether a connection is expressed with a bias of -3.0. The activation function of this output is a step function that outputs 1.0 when the input is greater than 0.0 and outputs 0.0 otherwise. In this way, the seed specifies that only connections that are local may be expressed (as in figure 2). Slight perturbations of the seed in the initial generation provide a variety of local connectivity patterns.

The alternative seed, which specifies locality along the x-axis only, is similar to the complete locality seed, except that the hidden nodes for Δy and Δz are absent. Finally, this x-locality seed structure is also inserted into both the low and high uniform threshold variants to determine whether a locality bias *alone* (i.e. when it is connected directly to the weight output) is sufficient to induce modularity, as opposed to biasing a separate LEO with locality.

All experiments are run with an implementation of HyperNEAT in C# by Phillip Verbancsics (available at http://eplex.cs.ucf.edu). Source code for the experiment is included in the same package. Because HyperNEAT is an extension of NEAT, most of the parameters are the same as in NEAT. The population size is 500. Available CPPN activation functions are absolute value, bipolar sigmoid, Gaussian,

linear, sine, and step. The probability of adding a node to the CPPN is 0.03 and the probability of adding a connection is 0.1. The probability of asexual reproduction is 0.75. The disjoint and excess node coefficients are both 1.0 and the weight difference coefficient is 1.0. The target number of species is 25. Finally, the elitism proportion is 0.1.

5. RESULTS

Each of the thresholding approaches are evaluated over 100 runs consisting of 5,000 generations of evolution. The performance is recorded for the champion of each generation as the percentage of the 256 patterns applied to the retinas that the ANN correctly classifies for both the left and right retina patterns (figure 4). As previously shown [1], standard HyperNEAT with any of the uniform thresholding approaches (unseeded or seeded) fails to achieve high average performance: The low threshold converges to 79% ($sd = 4\%$) correct on average and the high threshold reaches 82% ($sd = 8\%$) correct. Interestingly, both the locality-seeded standard-HyperNEAT with a uniform threshold and the dynamic distance threshold that might be assumed to encourage modularity *decrease* performance, reducing the average performance to 76% ($sd = 2\%$). The HyperNEAT-LEO without a locality seed increases performance to 84% ($sd = 8\%$), but does not significantly outperform the high uniform threshold without a locality seed. Only HyperNEAT-LEO with locality seeds significantly ($p < 0.001$ by Student's t-test) outperform all other approaches on average, reaching 95% ($sd = 7\%$) and 99% ($sd = 5\%$) accuracy, respectively. It is also interesting to note that an attempt to learn the same task with a direct encoding by Clune et al. [1] yielded only an average of 80% correct answers.

The differing performance of each approach can also be seen in how frequently they solve the problem perfectly (i.e. correctly classify 100% of the patterns), shown in figure 5. Of the 100 runs given to them, the uniform thresholds (low and high) solve the problem only 2% and 12% of the time, respectively. The locality-seeded standard-HyperNEAT and the dynamic distance threshold *never* find a perfect solution. Evolving the LEO without a seed solves the problem as often as the high uniform threshold, 12% of the time. The main result is that seeding the LEO with locality significantly improves the ability to find the perfect modular solution. The complete locality seed solves the problem 67% of the time and the seed that specifies locality along the x-axis increases the rate of finding the perfect solution to 91%. These results support the hypothesis that the CPPN itself should be given control of connectivity from a seed that begins evolution with significant locality.

A closer look at the structure of these final solutions gives insight into how and why they succeed or fail. For the uniform thresholds and the non-seeded HyperNEAT-LEO, the common outcome is to prefer fully-connected or near fully-connected patterns (figure 6). These patterns indicate that the struggle between finding modularity and establishing the correct pattern for each retina is dominated by optimizing the weight patterns, overriding movement towards modularity. Indeed, biasing the *weights* towards locality hinders the search for modularity by providing strong local optima. The distance threshold produces the opposite effect in its patterns (figure 7), leading to greater modularity yet less optimal weights. Only when locality is provided through seeded LEO does modularity become the prevailing pattern,

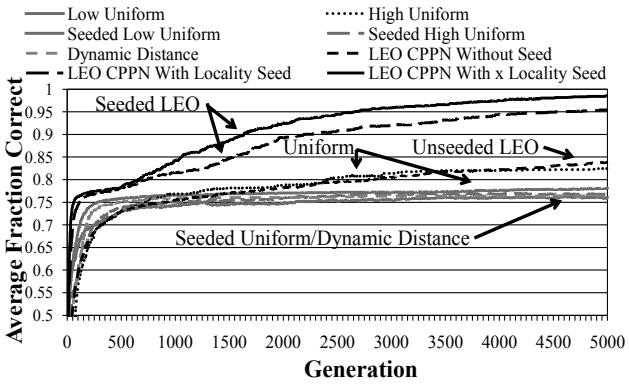

Figure 4: Average Percentage of Correct Classifications for Thresholding Methods. The fraction of correct classifications of the 256 patterns is averaged over 100 runs for each of the approaches. Each run lasts 5,000 generations and consists of a population of 500. The uniform and dynamic distance thresholds converge to sub-optimal values. The non-seeded LEO CPPN threshold is still improving at generation 5,000, but is not significantly outperforming the high uniform threshold. However, both the seeded LEOs exceed other approaches' performance with significance $p < 0.001$, demonstrating the importance of locality to the evolution of modularity.

while allowing the weights to be optimized separately (figure 8). Hence, clearly separated structures emerge consistently. Validating this inference, 64% of runs with the complete locality seed and 85% with the x-locality seed produce solutions with no crossing connections whatsoever between the left and right modules; all the other methods *fail* to produce such structures in more than 75% of runs.

6. DISCUSSION

Modularity is a key component in large and complex structures, in addition to regularity and hierarchy [13]. HyperNEAT is able to generate regular patterns through a hierarchy of function compositions, but has faced a challenge producing modular patterns [1]. By modifying HyperNEAT such that the expression of connectivity is independent of the pattern of the weights, modularity and regularity can be made independent from each other. Thus both of these key features can be optimized separately. Enhancing this idea, search can be biased by seeding evolution with geometric concepts, such as locality.

One way to understand why determining whether a connection is expressed should be independent from determining its weight is to consider how connection expression and weight determination might interact when they are both determined by the same variable, as in traditional HyperNEAT. In effect, the need to suppress connections in some parts of the network can lead to *distortions* in the pattern of weights, which are themselves a function of the network geometry. In struggling to balance these two demands through a single function, the CPPN ultimately succeeds at neither. This insight suggests that the traditional thresholding technique in HyperNEAT was probably not the best choice in retrospect, although it has been in use for several years [5, 8]. Nevertheless, the good news is that many interesting new structures may evolve now that this problem is uncovered.

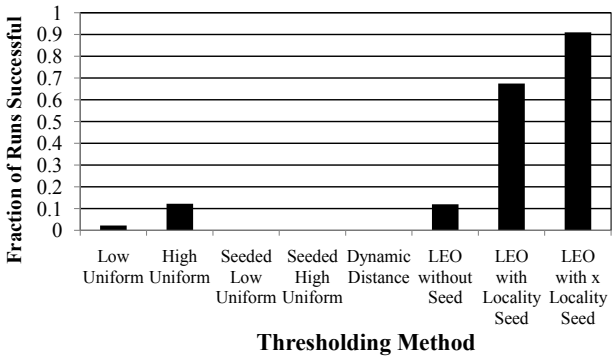

Figure 5: Fraction of Runs that are Perfect by Thresholding Method. The fraction of 100 runs that are successful in correctly classifying all of the 256 possible patterns is shown for each of the thresholding approaches after 5,000 generations. The seeded uniform and dynamic distance thresholds find the solution the least often, with success rates of 0%. The low uniform threshold is able to find a solution in 2% of the runs. Seeding non-LEO CPPNs with locality provides no extra benefit while the non-seeded LEO CPPN threshold matches the high uniform threshold; both find the solution in 12% of the runs. The global locality seed improves the odds of finding the solution to 67%, demonstrating how essential the concept of locality is not only to achieving a high fitness, but to finding the exact solution. The x-axis locality seed improves upon the locality seed, finding the solution in 91% of the runs, confirming the intuition that the solution's best opportunity for modularity in this domain is along the x-axis.

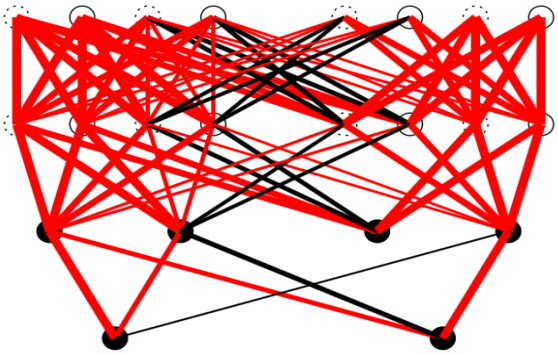

Figure 6: Typical Connectivity Pattern of Uniform Threshold. The uniform threshold fails to generate modularity. Instead, it consistently produces near fully-connected networks. These patterns are similar to those produced by the LEO without a seed. Optimizing weight patterns overrides the creation of modular structures. Red lines are negative weights, black line are positive weights, and line thickness indicates weight strength.

An exciting implication of this work is that both connectivity (i.e. which connections are expressed) and connection weights can be indirectly encoded as a *function of geometry* to provide key prerequisites to generating complex structures. The ability to manipulate the expression of connectivity independently of weights provides the ability to create

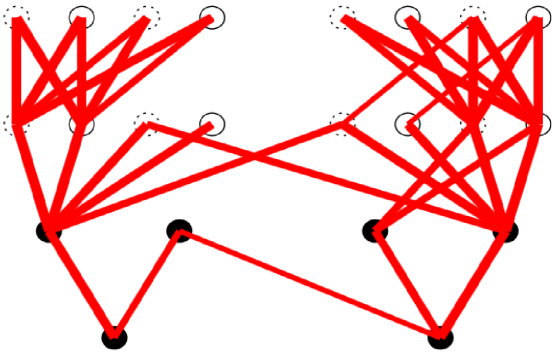

Figure 7: Typical Connectivity Pattern of Dynamic Distance Threshold. The dynamic distance threshold generates apparent modularity, but has difficulty finding the correct weight patterns. The problem is that the dynamic distance threshold impacts not only when connections are expressed, but the weight patterns that are created. Because longer distances mean higher thresholds, higher weight outputs become necessary, resulting in a movement towards extreme output values.

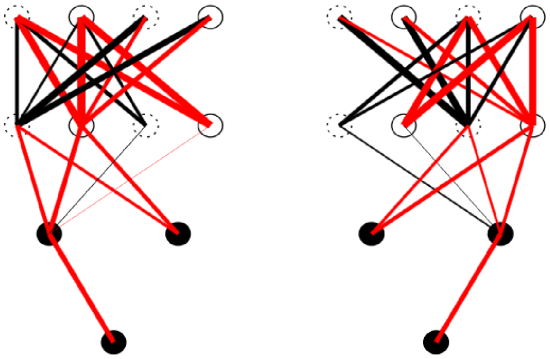

Figure 8: Typical Connectivity Pattern of LEO with Locality Seed. Modularity is commonly found when HyperNEAT-LEO is given the concept of locality. Once modularity is found, the regularities needed to solve the task for each module can be discovered in the weight pattern. This separation is possible because HyperNEAT-LEO specifies the pattern of weights and the pattern of connection expression through different outputs of the CPPN. Thus both can be evolved independently, freeing evolution from the burden of searching for a weight pattern that has both the correct weight settings to solve the problem and the necessary modularity.

modules. However, modularity itself follows important geometric principles. One of these, locality, is inherent in the natural universe, that is, components that are grouped into a module are necessarily located near each other because of physical constraints. Simulated structures do not necessarily have such constraints placed upon them, but the principles that arise because of such constraints can be helpful in creating structures that resemble those of nature.

Interestingly, the need for an initial bias towards locality to consistently solve the problem also suggests that the path to encoding locality is inherently deceptive with respect to the fitness function in this task. Such deception may turn

out common when geometric principles such as locality that are conceptually orthogonal to the main objective are nevertheless essential to achieving it consistently. Thus seeding with the right geometric bias may prove an important tool to avert deception in many domains.

Importantly, it is not enough simply to enforce the concept of locality or bias the structure toward locality to create modules, as demonstrated by the failure of the dynamic distance threshold and the locality-seeded standard HyperNEAT. Instead, evolution should be allowed to find its own separate rules for modularity. This insight gives hope that HyperNEAT can now be applied to new domains that may benefit from modular neural organization.

7. CONCLUSION

This paper investigated variations on thresholding connectivity in HyperNEAT, introducing the dynamic distance threshold and Link Expression Output. While HyperNEAT has struggled with modularity in the past, decoupling the weight pattern from whether a connection is expressed facilitated the ability to generate modular ANN weight patterns with HyperNEAT-LEO. Instead of failing to find the solution to the modular Retina Problem, HyperNEAT-LEO with a locality seed is almost always able to find the perfect solution, whose modularity is visually apparent in the network geometry. Thus an extension of HyperNEAT now potentially captures two key features of natural systems: regularities and modularity.

Acknowledgements

This research is supported in part by a Science, Mathematics, and Research for Transformation (SMART) fellowship from the American Society for Engineering Education (ASEE) and Naval Postgraduate School.

8. REFERENCES

[1] Jeff Clune, Benjamin E. Beckmann, Philip K. McKinley, and Charles Ofria. Investigating whether hyperneat produces modular neural networks. In *Proc. of the Genetic and Ev. Comp. Conf.*, New York, NY, USA, 2010. ACM Press.

[2] Jeff Clune, Benjamin E. Beckmann, Charles Ofria, and Robert T. Pennock. Evolving coordinated quadruped gaits with the hyperneat generative encoding. In *Proc. of the IEEE Congress on Ev. Comp. Special Section on Ev. Robotics*, Piscataway, NJ, USA, 2009. IEEE Press.

[3] Jeff Clune, Kenneth O. Stanley, Robert T. Pennock, and Charles Ofria. On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Ev. Comp.*, 2011.

[4] David D'Ambroiso and Kenneth O. Stanley. Evolving policy geometry for scalable multiagent learning. In *Proc. of the 9th Int. Conf. on Auto. Agents and Multiagent Sys.*, page 8, New York, NY, USA, 2010. ACM Press.

[5] David D'Ambrosio and Kenneth O. Stanley. A novel generative encoding for exploiting neural network sensor and output geometry. In *Proc. of the Genetic and Ev. Comp. Conf.*, New York, NY, 2007. ACM Press.

[6] David B. D'Ambrosio and Kenneth O. Stanley. Generative encoding for multiagent learning. In *Proc. of the Genetic and Ev. Comp. Conf.*, New York, NY, 2008. ACM Press.

[7] Ivan Garibay, Ozlem Garibay, and Annie Wu. Effects of module encapsulation in repetitively modular genotypes on the search space. In *Proc. of the Genetic and Ev. Comp. Conf.*, pages 1125–1137, New York, NY, USA, July 2004. ACM.

[8] Jason Gauci and Kenneth O. Stanley. Generating large-scale neural networks through discovering geometric regularities. In *Proc. of the Genetic and Ev. Comp. Conf.*, New York, NY, 2007. ACM Press.

[9] Jason Gauci and Kenneth O. Stanley. A case study on the critical role of geometric regularity in machine learning. In *Proc. of the 23d AAAI Conf. on AI*, Menlo Park, CA, 2008. AAAI Press.

[10] Jason Gauci and Kenneth O. Stanley. Autonomous evolution of topographic regularities in artificial neural networks. *Neural Computation*, page 38, 2010.

[11] Leland H. Hartwell, John H. Hopfield, Stanislas Leibler, and Andrew W. Murray. From molecular to modular cell biology. *Nature*, 402, 1999.

[12] Nadav Kashtan and Uri Alon. Spontaneous evolution of modularity and network motifs. *Proc. of the NAS*, 102(39):13773–13778, September 27 2005.

[13] Hod Lipson. Principles of modularity, regularity, and hierarchy for scalable systems. *Journal of Biological Physics and Chemistry*, 7, 2007.

[14] Kenneth O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines Special Issue on Dev. Sys.*, 8(2):131–162, 2007.

[15] Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. A hypercube-based indirect encoding for evolving large-scale neural networks. *Artificial Life*, 15, 2009.

[16] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.

[17] Kenneth O. Stanley and Risto Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130, 2003.

[18] Kenneth O. Stanley and Risto Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100, 2004.

[19] Matthew E. Taylor, Shimon Whiteson, and Peter Stone. Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In *Proc. of the Genetic and Ev. Comp. Conf.*, pages 1321–1328, New York, NY, July 2006. ACM Press.

[20] Alan M. Turing. The Chemical Basis of Morphogenesis. *Royal Society of London Philosophical Transactions Series B*, 237:37–72, August 1952.

[21] Phillip Verbancsics and Kenneth O. Stanley. Evolving static representations for task transfer. *Journal of Machine Learning Research*, 11, 2010.

[22] Gunter P. Wagner and Lee Altenberg. Complex adapatations and the evolution of evolovability. *Evolution*, 50, 1996.

[23] Shimon Whiteson. Improving reinforcement learning function approximators via neuroevolution. In *Proc. of the 4th Int. Conf. on Auto. Agents and Multiagent Sys.*, pages 1386–1386, New York, NY, USA, 2005. ACM.

[24] Shimon Whiteson and Daniel Whiteson. Stochastic optimization for collision selection in high energy physics. In *Proc. of the 19th Annual Innov. Apps. of AI Conf.* , Vancouver, British Columbia, Canada, July 2007. AAAI Press.

[25] R. Paul Wiegand, Gautham Anil, Ivan Garibay, Ozlem Garibay, and Annie Wu. On the performance effects of unbiased module encapsulation. In *Proc. of the Genetic and Ev. Comp. Conf.*, pages 1729–1736, New York, NY, USA, July 2009. ACM.