Investigating Whether HyperNEAT Produces Modular Neural Networks

Jeff Clune, Benjamin E. Beckmann, Philip K. McKinley, and Charles Ofria
Department of Computer Science and Engineering, Michigan State University
East Lansing, MI, 48824, USA
jclune@msu.edu, beckma24@msu.edu, mckinley@cse.msu.edu, ofria@msu.edu

ABSTRACT

HyperNEAT represents a class of neuroevolutionary algorithms that captures some of the power of natural development with a computationally efficient high-level abstraction of development. This class of algorithms is intended to provide many of the desirable properties produced in biological phenotypes by natural developmental processes, such as regularity, modularity and hierarchy. While it has been previously shown that HyperNEAT produces *regular* artificial neural network (ANN) phenotypes, in this paper we investigated the open question of whether HyperNEAT can produce *modular* ANNs. We conducted such research on problems where modularity should be beneficial, and found that HyperNEAT failed to generate modular ANNs. We then imposed modularity on HyperNEAT's phenotypes and its performance improved, demonstrating that modularity increases performance on this problem. We next tested two techniques to encourage modularity in HyperNEAT, but did not observe an increase in either modularity or performance. Finally, we conducted tests on a simpler problem that requires modularity and found that HyperNEAT was able to rapidly produce modular solutions that solved the problem. We therefore present the first documented case of HyperNEAT producing a modular phenotype, but our inability to encourage modularity on harder problems where modularity would have been beneficial suggests that more work is needed to increase the likelihood that HyperNEAT and similar algorithms produce modular ANNs in response to challenging, decomposable problems.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning: Concept Learning, Connectionism and Neural Nets

General Terms: Experimentation, Algorithms

Keywords: HyperNEAT, NEAT, Neuroevolution, Modularity, Generative Encodings, Developmental Encodings, Indirect Encodings, Artificial Neural Networks.

1. INTRODUCTION AND BACKGROUND

A long-term goal in the fields of evolutionary computation, neuroevolution, and artificial life is to synthetically evolve phenotypes as complicated as those seen in the natural world. Many complex natural organisms exhibit modularity, regularity, and hierarchy [9, 14, 20, 25, 26], which increase the evolvability of these organisms [10, 13, 17, 19]. *Modularity* is the localization of function within an encapsulated unit, which in a network entails clusters of nodes with high connectivity within the cluster and low connectivity to nodes outside the cluster [13, 17]. *Regularity* refers to the compressibility of the information describing a structure, and typically involves symmetries and module repetition [17]. *Hierarchy* is the recursive composition of lower-level units [17]. Note that modularity does not require regularity, as is often assumed: the single wheel on a unicycle is a module, whereas the four wheels on a car are a regular repetition of a wheel module [17]. Without the ability to evolve phenotypes that possess these characteristics, it may be difficult to synthetically evolve creatures as complicated as those found in nature [1, 17, 20]. Modularity is especially important for neural networks, where it can improve both evolvability and learning, because modular networks can more easily be rearranged to produce new functions [13, 19]. These benefits likely explain why natural brains display a high degree of modularity, regularity, and hierarchy [9, 14, 20, 25, 26]. Designs engineered by humans also possess these properties for the same reasons: they make it easier to design and modify complex artifacts.

Modularity, regularity, and hierarchy arise in natural organisms as a result of a complex developmental process [1, 20]. A desire to produce these design principles in synthetic evolution has led many researchers to switch from *direct encodings*, where each phenotypic element is specified by a corresponding genomic element, to *generative encodings* that resemble natural developmental processes, wherein elements in a genome can influence many parts of a phenotype [22]. It has been shown that generative encodings are capable of producing modularity, regularity, and hierarchy in phenotypes [10], and specifically can create regularity and modularity in evolved neural networks [8, 11]. These generative encodings are based on rewriting symbols, such as Lindenmayer Systems [10-12, 16], or programs that are recursively called at vertices in a graph [8]. These representations perform well in part because they strongly and explicitly bias evolution towards phenotypes with modularity, regularity, and hierarchy [10].

A different type of generative encoding, called a Compositional Pattern Producing Network (CPPN), includes a generative process that abstracts how natural organisms develop complexity in a novel way [1, 23, 24]. CPPNs have shown promise as an evolutionary encoding, but they were not designed to generate modularity, regularity, and hierarchy as explicitly as previous generative encodings. It is therefore important to determine the degree to which CPPNs produce these properties in their phenotypes.

A CPPN is a high-level abstraction of biological genetic regulatory networks, which construct positional information that determines the fate of phenotypic elements in organisms. While

GECCO'10, July 7–11, 2010, Portland, Oregon, USA.

CPPNs have an artificial component that provides each phenotypic element with its location in an *objective* coordinate space, natural development goes through intermediate steps to arrive at a similar result [1]. CPPNs also abstract away growth through intermediate forms, and instead build the 'adult' phenotype directly. Despite these differences with natural development, CPPNs do capture a key factor that enables natural development to produce complexity: determining the fate of phenotypic elements as a function of their location in complex geometric coordinate frames [1].

Images evolved via artificial selection with the CPPN encoding look complex and natural [23, 27]. These images can easily be selected to resemble animals and human artifacts. They are often highly regular, including symmetries and repeated themes, with and without variation. These images suggest that CPPNs present a promising encoding that is on the path toward synthetically evolving complex phenotypes that resemble natural organisms.

CPPNs can encode neural networks via the HyperNEAT algorithm [24], which is described in Section 2. HyperNEAT has performed well on a wide range of problems, such as generating gaits for legged robots [3], pattern recognition [24], controlling simple multi-agent systems [6], and evaluating checkers boards [7]. Because of HyperNEAT's potential as an effective neuroevolutionary algorithm, and given that HyperNEAT captures some of the power of biological development, it is worthwhile to investigate whether HyperNEAT tends to produce ANNs that exhibit modularity, regularity, and hierarchy. It has already been shown that HyperNEAT produces *regular* ANNs that exploit the regularity of problems [3, 4, 7]. This paper investigates the previously unstudied question of whether HyperNEAT produces *modular* ANNs. In future investigations we will address whether HyperNEAT can produce *hierarchical* ANNs. We would like to emphasize that in this paper we focus on modularity in evolved *phenotypes*. In future work we also plan to investigate the modularity, regularity, and hierarchy of HyperNEAT *genotypes*.

We tested whether HyperNEAT and a direct encoding control produce modular ANNs on a problem that has previously been shown by Kashtan and Alon [13] to generate modular ANNs with a different direct encoding neuroevolution algorithm. In contrast to those results, this problem did not encourage modularity in the direct encoding we tested, raising a question about the generality of Kashtan and Alon's results. We also found that HyperNEAT performed poorly on this problem, and variants of it, and did not produce modular ANNs. We then tested whether HyperNEAT would have done better had it produced a modular ANN by imposing modularity on its ANN phenotypes. With this imposed

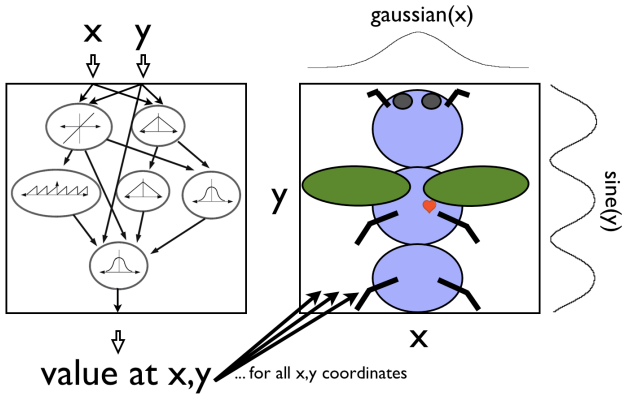

Figure 1. CPPNs can compose math functions to generate the properties of symmetry and modular repetition, with and without variation. This figure is adapted from [23].

modularity, HyperNEAT's performance improved. These results show that, irrespective of how the direct encoding performed on this problem, HyperNEAT would have done better had it produced modular ANNs. We next tested two techniques to encourage HyperNEAT to produce modularity automatically, but did not observe an increase in either modularity or performance. Finally, we conducted tests on a simplified version of the problem and found that HyperNEAT quickly was able to produce modular solutions that solved the problem. We therefore present the first documented case of HyperNEAT generating a modular phenotype, but our inability to encourage modularity on harder problems where modularity would have increased performance suggests that more work is needed to increase the likelihood that HyperNEAT and similar algorithms will produce modular ANNs in response to challenging, decomposable problems.

2. THE HYPERNEAT ALGORITHM AND A DIRECT ENCODING CONTROL

HyperNEAT [24] is a generative encoding that evolves ANNs with the principles of the widely used NeuroEvolution of Augmenting Topologies (NEAT) algorithm, which is described below [21]. HyperNEAT evolves Compositional Pattern Producing Networks (CPPNs) [23], each of which is a mathematical function. For example, to evolve two-dimensional pictures [27], the inputs to a CPPN could be the Cartesian coordinates of each pixel on a canvas. The CPPN output could determine the color of each pixel (Figure 1).

Evolution modifies a population of CPPNs. Each CPPN is a directed network, where each node is itself a mathematical function. The nature of the functions included can enable a variety of desirable properties, such as symmetry (e.g., a Gaussian function) and repetition (e.g., a sine function) that evolution can take advantage of. Nested coordinate frames can develop in the CPPN. For instance, a sine function early in a network can create a repeating theme that, when passed into the symmetric Gaussian function, creates a repeating, symmetric motif, as demonstrated by the body segments in Figure 1. This process is similar to how natural organisms develop [1]. For example, many organisms set up a repeating coordinate frame (e.g., body segments) within which are symmetric coordinate frames (e.g., left-right body symmetry). Asymmetries can be generated by referencing global coordinate frames, such as the x-axis. The links that connect and allow information to flow between nodes in a CPPN have a weight value that can magnify or diminish the values that pass along them. Mutations that change these weights may, for example, give a stronger influence to a symmetry-generating part of a network while diminishing the contribution from another part.

When CPPNs are evolved to generate ANNs, the algorithm is called HyperNEAT [24]. In this case, the inputs to the CPPN are the locations in three-dimensional Cartesian space of both the source and target nodes of each link in a target ANN, and a constant bias value. The function takes these seven values (x_1, y_1, z_1, x_2, y_2, z_2, bias) as inputs and produces an output value that determines the weight of the link between the associated source and target nodes in the ANN. All source and target nodes for each link in the ANN are iteratively passed as inputs to a CPPN to determine the weight of each link. Thus, a CPPN is a genome that encodes for an ANN phenotype [24].

One benefit of HyperNEAT is that it can exploit the geometry of a problem [5, 24]. Because the link values between nodes in the final ANN substrate are a function of the geometric positions of those nodes, if those geometric positions represent aspects of the

problem that are relevant to its solution, then HyperNEAT can exploit such information. For example, when playing checkers, the concept of adjacency (on the diagonals) is important. Link values between adjacent squares may need to be different than link values between distant squares. HyperNEAT can create this sort of connectivity motif and repeat it across the board [7, 24]. In the case of quadruped locomotion, HyperNEAT exploited geometric regularities to generate front-back and left-right symmetries to produce common gaits [3, 5].

Variation in HyperNEAT occurs when mutations or crossover alter a CPPN. Mutations can add a node, which results in the addition of a function to a CPPN, or change its link weights. The functions in CPPNs in this paper are the standard set: sine, sigmoid, Gaussian, and linear [24]. A population of CPPN networks is evolved with the NEAT algorithm, which was originally designed to evolve ANNs. NEAT can be effectively applied to CPPNs because CPPN networks are similar in structure to ANNs.

The NEAT algorithm contains three key elements [21]. Initially, it starts with small genomes that encode simple networks and slowly *complexifies* them via mutations that add nodes and links to the network. This complexification enables the algorithm to evolve the network topology in addition to its link weights. Secondly, NEAT has a fitness sharing mechanism that preserves diversity and gives new innovations time to be tuned by evolution before competing them against more mature rivals. Finally, NEAT tracks historical information to perform crossover in a way that is effective, yet avoids the need for expensive topological analysis. A full explanation of NEAT can be found in Stanley & Miikkulainen [21].

A direct encoding control for HyperNEAT is FT-NEAT [2, 4, 24]. FT-NEAT independently evolves each weight in the ANN and does not add hidden nodes (complexify). All other elements from NEAT (e.g., its crossover and diversity preservation mechanisms) remain the same between HyperNEAT and FT-NEAT.

The specific parameters for these experiments are similar to those of previous HyperNEAT studies [24], and can be found at http://devolab.msu.edu/SupportDocs/HyperNEATModularity.

3. THE RETINA PROBLEM

An informative test of whether HyperNEAT produces modular ANNs is to try it on a problem where modularity is known to be helpful, and in an environmental regime that has been shown to encourage modularity in a neuroevolution algorithm. Fortunately, previous research has been conducted on such a problem [13]. Kashtan and Alon demonstrated that environmental regimes that switch between problems with *modularly varying goals* (MVG) increase the evolution of modular phenotypic networks. MVG environments switch between tasks that have shared subproblems, but where the overall problem is solved by combining answers to these subproblems in different ways. On two different problems, Kashtan and Alon demonstrate that MVG environments produce highly modular networks. They also show that *fixed goal* (FG) controls that evolve to solve a single unchanging problem produce non-modular networks, even though the fixed goal was identical to one of goals from the MVG regime and thus had the same subproblems. The MVG treatments also solved problems in an order of magnitude fewer generations. Moreover, the evolved modules of the MVG networks solved the subproblems Kashtan and Alon had designed into the overall problems. Over time, solutions evolved that allowed the modules to be reconfigured via a single or small number of mutations, thereby enabling quick adaptations from one environment to another. In subsequent work it was shown that, after an environmental change, modular networks were faster at adapting both to previously seen and novel environments: This ability to quickly adapt to new environments is made easier because of the modularity that evolved in the networks [19]. Inspired by these findings, scientists tested and confirmed that similar results hold for natural organisms: bacteria that live in changing environments have more modular metabolic networks [15, 18].

Kashtan and Alon's results are consistent with our expectations for when modularity is useful. Modularity is not necessarily helpful, and may be harmful, when designing a solution for a single, unchanging problem [17]. Modularity becomes beneficial when designs need to be changed quickly, because modules that solve subproblems can be easily reorganized [17].

Kashtan and Alon's first problem involved evolving the connections of networks of NAND gates to solve Boolean logic functions. Their second problem consisted of evolving neural networks to perform pattern recognition. We chose their second problem as the test problem in this paper because HyperNEAT was designed to evolve neural networks. The second problem evolves a neural network to separately recognize patterns, or 'objects', on the left and right sides of an artificial retina (Figure 2a). The retina consists of eight pixels, four per side, which were the inputs to a neural network with sigmoid activation functions. The left four pixels (the left pane) can form 16 unique patterns, half of which are considered Left Objects. The same is true for the right four pixels (the right pane). The goal is to have the single output of the network answer one of two Boolean logic questions: [L AND R] (true if there is a Left Object and a Right Object), or [L OR R] (true if there is a Left Object, if there is a Right Object, or both). This *Retina Problem* is challenging because the network must independently recognize and process low-level patterns before processing that

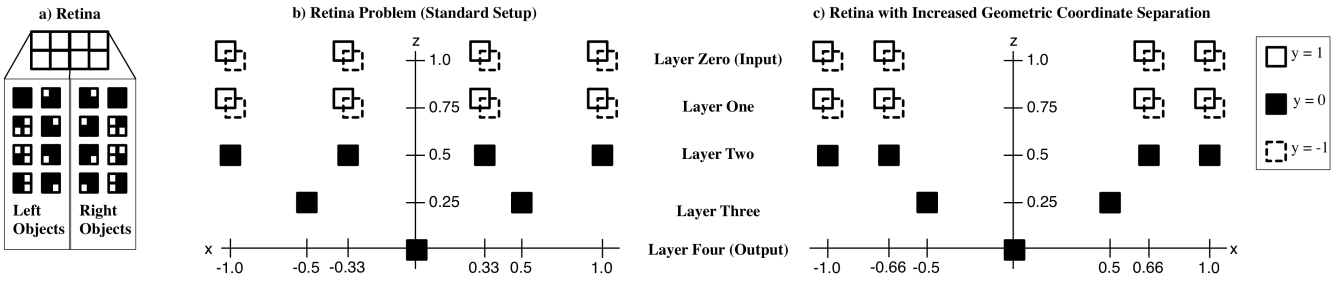

Figure 2. (a) The eight-pixel artificial retina and the patterns that constitute Left and Right Objects (adapted from [13]). (b) The geometric representation of the ANN nodes for the Standard Setup of the Retina Problem. (c) The geometric representation for the Retina Problem with Increased Geometric Coordinate Separation. The x, y, and z coordinate values for each node are passed into the CPPN when determining the weight of links between nodes.

information to determine if a higher-level pattern is present [13]. The networks had four feed-forward layers in addition to an input layer (Figure 2b-c). Layer 1, which received connections from the input layer, had eight input neurons. Layers 2 and 3 were hidden layers with four and two neurons, respectively. The output layer had a single neuron.

A human engineer immediately recognizes the modularity in the Retina Problem: The left and right panes can be processed independently to determine if an object is present. The information can then be combined in either a logical AND or OR. There are non-modular ways that may be equally good at solving either problem, but such non-modularity will likely make it more difficult to switch from a network that solves one problem to a network that solves the other, where difficulty is measured by the number and magnitude of link weight changes that need to be made.

We implemented this problem in a version of the HyperNEAT software that was used in several previous studies [2-5, 7, 24]. Pixels were limited to 'on' and 'off' states, represented as input values of 3.0 and -3.0, respectively. A bias neuron with a constant input of 3.0 had evolvable connections to all neurons. This feature serves a similar function to the evolvable thresholds in Kashtan and Alon's setup [13]. Outputs were considered true if they were close to 1 and false if they were close to -1. Our fitness function was inversely proportional to the difference (the error) between the correct answer (1 or -1) and the network output. Specifically, the fitness function summed the error across all 256 possible input patterns and squared the result to magnify the importance of slight improvements.

The specifics of our implementation differ in certain ways from Kashtan and Alon's [13]. While the description of their model is not complete, it appears that their inputs and outputs were binary, the activation functions of their neurons were step functions with only three possible thresholds, and their link weights consisted of a small set of discrete values. They evolved their networks via a standard direct encoding genetic algorithm with mutation and crossover. Their fitness was a function of the percent of correct answers provided across 100 randomly chosen input patterns. These differences, while seemingly minor, may explain the different qualitative results we observe from those of Kashtan and Alon [13].

Kashtan and Alon evolved networks in an FG regime [L AND R] and an MVG regime, wherein the rewarded task switched every 20 generations from [L AND R] to [L OR R]. They continued each evolutionary run until the networks output the correct answer for 95% of the input patterns, at which point they considered the problem solved. That took a median of 21,000 generations in the FG regime, which was nearly an order of magnitude slower than in the MVG regime, which took 2,800 generations. The MVG networks were more modular, and could adapt to an environmental change from one goal to the other in about 3 generations, often via a single mutation [13].

4. RESULTS AND INTERPRETATION

4.1 Retina Problem

We tested the performance of HyperNEAT and FT-NEAT on the Retina Problem for two MVG regimes, one that alternated between tasks [L AND R] and [L OR R] every 20 generations (*MVG-20*), which was the rate used by Kashtan and Alon, and another that alternated every 100 generations (*MVG-100*). We also conducted experiments with faster and slower rates of change, but the results were not qualitatively different (data not shown). We tested two FG regimes (*FG-AND* and *FG-OR*), one for each of the tasks. For each experimental treatment discussed in this paper we performed 20 runs of evolution with different random number generator seeds, and report the median (bold lines) and 25th and 75th percentiles (thin lines). To represent fitness values, we show the percent of test cases the best organism in the population provided the correct answer for. The nature of the logic functions means that always outputting 1 (in the OR environment) or 0 (in the AND environment) achieves a score of 75%. For this reason, it was rare to see the best organism in each generation score below 75%. Each run had a population size of 500, which is large for HyperNEAT experiments [24].

The results, presented in Figure 3a, reveal that HyperNEAT does not perform well on this problem. Recall that Kashtan and Alon's direct encoding achieved 95% accuracy in both the FG and MVG regimes. FT-NEAT also performed poorly (Figure 4), and its results were qualitatively the same as for HyperNEAT on the FG-AND, FG-OR, and MVG treatments. While Kashtan and Alon did perform evolution for many more generations, additional experiments up to 30,000 generations for both HyperNEAT and FT-NEAT revealed that longer experiments do not change the qualitative results (data not shown). More likely, the difference in absolute success has to do with the differences in the neural networks. However, alternative experiments with different selective pressures produced networks that perfectly solved the FG problems, suggesting that the difference between our results and those of Kashtan and Alon is not due to any limitation in the capability of the networks in our experiments, but is instead related to differing evolvability between the configurations.

To better understand why HyperNEAT performed poorly, we ran the same experiment, with the same number of nodes and potential links in the ANN, but where ANNs were rewarded for

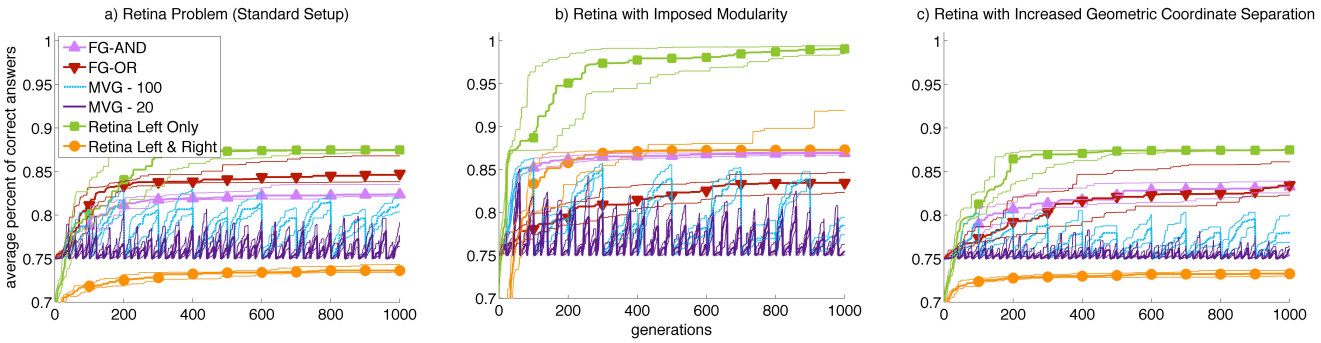

Figure 3. Performance versus evolutionary time for HyperNEAT on (a) the Retina Problem (Standard Setup), (b) the Retina Problem with Imposed Modularity, and (c) the Retina Problem with Increased Geometric Coordinate Separation. Plotted is the percent of the 256 trials that the network output the correct answer. Medians are shown as bold lines surrounded by the 75th and 25th percentiles of the data. See the text for explanations of what constituted a correct answer for different variants of the problems.

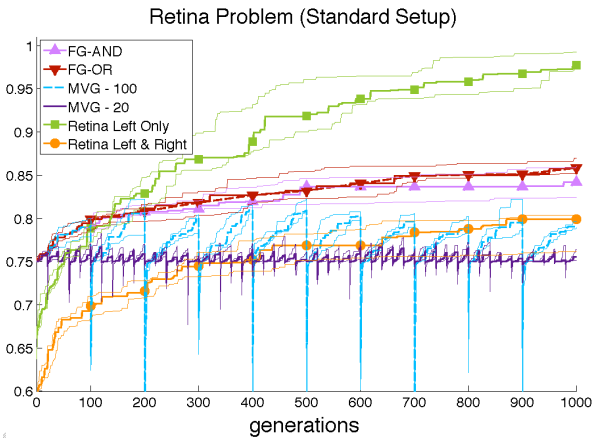

Figure 4. The performance of FT-NEAT on the Standard Setup of the Retina Problem. Medians are shown as bold lines surrounded by the 75th and 25th percentiles of the data. Note that the y-axis scale is different than in Figure 3.

correctly identifying only Left Objects in the FG-AND problem. The evaluated output was taken from the left node of layer 3 (the layer just before the output layer). In this easier version of the problem, which we call *Retina Left Only*, HyperNEAT performed better, but still had difficulties (Figure 3a). These difficulties may have occurred because HyperNEAT created too many links in its substrate and therefore did not ignore the inputs from the right panel when identifying Left Objects. FT-NEAT, on the other hand, performed significantly better, with 17 of 20 treatments surpassing 95% accuracy ($p < 0.001$, Mann-Whitney U rank test, Figure 4).

We conducted a similar experiment, but rewarded networks that could correctly report whether there were Left Objects and Right Objects, respectively, in the left and right nodes of layer 3. This is a particularly illuminating experiment because a network must be able to first solve this task before computing the logical AND or OR of these answers, which is required for the optimal solution to the FG and MVG problems. The results of this experiment, which we call *Retina Left & Right*, demonstrate that HyperNEAT did worse on this task than all other versions of the problem (Figure 3a). That HyperNEAT was unable to independently determine the presence or absence of Left Objects and Right Objects helps explain why it did not do well on the harder tasks that require further processing this information. Of particular interest is how much better the FG-AND and FG-OR treatments performed than the Left & Right treatment. Given that HyperNEAT had difficulty independently identifying Left and Right Objects, we can infer that the strategy it employed on the FG treatments to perform better than the Left & Right treatment did not independently process the left and right panels. HyperNEAT likely took advantage of the locally optimal shortcut of always outputting 1s or 0s, which yields fitness values of 75%. It could then have increased its performance up to the level it achieved by encoding some additional information about the problem, such as certain situations in which to provide the other output.

More important than the absolute difference between Kashtan & Alon's results and those of HyperNEAT and FT-NEAT is the qualitative difference: the MVG regimes performed worse than the FG-AND regime, which was the opposite of what occurred in Kashtan and Alon's study. Our result raises questions as to the generality of Kashtan and Alon's discovery that environments with MVG will generate the evolution of modular networks. While there are differences between Kashtan and Alon's experimental setups and our own, the differences are relatively small and should not preclude such a seemingly general result. We will investigate what differences in the implementations led to the differing results in future work.

Despite the differences between our results and those of Kashtan and Alon, the Retina Problem still serves as a diagnostic problem regarding the ability of an algorithm to produce modularity. Because the problem can be decomposed on the left and right sides until the final layer, networks that are more modular should perform better. The case is even clearer for the Retina Left & Right problem, because the left problem is independent of the right problem. For the remainder of the paper, we utilize the Retina Problem and variants of it to investigate HyperNEAT's ability to generate modular ANNs.

We hypothesized that HyperNEAT may have performed poorly because it was not producing modular ANNs. To test whether HyperNEAT's ANNs were modular, we counted the number of active (non-zero) links in the substrate. A modular design that processed the left and right panels separately before combining them has at most 69 of the possible 121 links in the neural network, or 57%. This number is small because there are no links between the two modules. The degree to which the percent of links in a network is over 57% suggests the extent to which the ANN is interconnected instead of modular. This link-counting measure is a crude estimation of modularity, but it is not accurate to simply count the links between the nodes on the left and right side of the coordinate space, because it is possible for evolution to create modules that are not correlated with geographic location. In future work we plan to quantify the modularity of these networks with a more sophisticated modularity metric.

The FG regimes had a median of 94% and 96% of links active (SD 6%), respectively, for the FG-AND and FG-OR treatments. The MVG-100 and MVG-20 regimes had medians of 94% and 95% (SD 3%, 7%). For many runs, 100% of the links in the champion ANN were active, which is the lowest level of modularity possible. There were no statistically significant differences in the link percentages between the FG and MVG treatments ($p > 0.05$, Mann-Whitney U rank test). We conclude from these data that part of the reason HyperNEAT performed poorly on both the FG and MVG tasks is because it has difficulty turning off links and thus produced ANNs with low levels of modularity.

4.2 Retina Problem with Imposed Modularity

We next investigated whether HyperNEAT would have done better had it discovered the left-right modularity of the problem. To test this, we disabled all connections between the left and right sides of the network, except between layers 3 and 4. Disabling of cross-links ensured that information from the left and right panels was processed independently until it was combined in the final layer.

This *Imposed Modularity* treatment improved the performance of every treatment (Figure 3b, $p < 0.001$ comparing the fitnesses of the generation champions per treatment from the final generation with a Mann-Whitney U rank test) except for FG-OR, which performed worse ($p < 0.05$). The decline in FG-OR performance with imposed modularity, while slight, is counterintuitive and was anomalous compared to the results from the other treatments. We hypothesized that this odd result may have occurred only because evolution had not yet leveled off, which was more so the case for the FG-OR treatments than the others. To test whether additional generations would make a difference, we extended the FG-OR experiments to 3000 generations, at which point the Imposed Modularity treatment outperformed the Standard Setup (the setup

with results plotted in Figure 3a), although the difference was not statistically significant ($p > 0.05$, Mann-Whitney U rank test). Four of the extended FG-OR with Imposed Modularity runs reached at least 95% accuracy, further demonstrating that HyperNEAT is capable of solving the Retina Problem with the neural networks used in this paper. None of the extended FG-OR runs without imposed modularity reached a fitness level of greater than 90%. It is not obvious why Imposed Modularity is less helpful on the FG-OR treatment than in the other treatments: It may be that the imposed modularity is interfering with the exploitation of a locally optimal strategy that is being used in the Standard Setup. Kashtan and Alon did not report experimenting with FG-OR, so we cannot compare our results to theirs [13].

These results confirm that HyperNEAT would have done better had it generated a modular network that independently processed the left and right panels. Interestingly, the largest effects were in two non-MVG treatments. In the Retina Just Left treatment, HyperNEAT scored nearly perfectly with imposed modularity (Figure 3b). This result demonstrates that the subproblems of identifying Left and Right Objects are not impossible for HyperNEAT to solve (experiments focusing just on the right side were qualitatively similar, data not shown). The imposed modularity also substantially improved the performance of the Retina Left & Right treatment. Four of these treatments achieved scores above 95%, with one at 99%, and none scored below 86%. Both the Retina Left Only and Retina Left & Right problems are thus demonstrations of problems where modularity is beneficial, but where HyperNEAT did not discover such modularity on its own. These results emphasize that HyperNEAT would perform better on some FG problems if it were better able to create modular ANNs. This is not to say that modularity is necessary, but just that in this case it improves the likelihood of evolving a high quality solution. That four of the Left & Right treatments scored above 95% also offers additional evidence that HyperNEAT is capable of solving the Retina Problem with the neural networks used in this paper, because putting this information together into an AND or an OR function is possible in this setup.

That imposed modularity increased performance on the Left & Right problem may help us infer why imposed modularity aided the performance of HyperNEAT on the FG-AND problem. The similarity in fitness scores between the Left & Right treatment and the FG-AND treatment with imposed modularity may indicate that the FG-AND treatment with imposed modularity did implement the globally optimal strategy of independently processing the left and right panels, albeit in an imperfect way. If so, this would be an interesting demonstration of how modularity helped evolution switch from a locally optimal strategy to a higher-performing and possibly globally optimal strategy. Unfortunately, it is difficult to determine if these networks did indeed correctly process the left and right panels by recording the values at the associated nodes in layer 3, because evolution can internally represent information in different ways.

4.3 Retina with Fewer Links

One factor that may hamper HyperNEAT's ability to create modular ANNs is that it produces too many substrate links. To test this hypothesis, we increased the range of CPPN outputs that were converted to an ANN link weight of zero, which effectively eliminates the link. For the previous experiments, CPPN outputs in the range of -0.1 to 0.1 resulted in ANN links of 0. Such a range was built into HyperNEAT to facilitate the elimination links in its ANN phenotypes [24]. We call this parameter *ZeroOutHalfWidth*. As the ZeroOutHalfWidth increases, a wider range of CPPN output values result in ANN link weights of zero, decreasing the expected number of ANN links.

We tested ZeroOutHalfWidth values of 0.2, 0.4, 0.6, 0.8, 0.9, 0.95 and 0.99, and compared the results to the default value of 0.1 (Figure 5). Altering the ZeroOutHalfWidth parameter had little effect on fitness and did not raise fitness up to the levels observed with imposed modularity. All of the treatments with different ZeroOutHalfWidth values, but without imposed modularity, were significantly worse than the Imposed Modularity treatment on both the FG-AND and Left & Right problems ($p < 0.001$, Mann-Whitney U rank test).

The data reveal that higher ZeroOutHalfWidth values did reduce the number of ANN links in both problems (Figure 5, bottom row). Importantly, the number of ANN links for some of the different ZeroOutHalfWidth values were roughly similar to the number of links in the Imposed Modularity treatment. Given that these treatments had similar numbers of ANN links, but performed significantly worse than the Imposed Modularity treatment, it is likely that the resulting networks were not very modular. This result suggests that it is not merely the inability to eliminate ANN links that prevents HyperNEAT from discovering modular solutions to these problems, but that HyperNEAT also has trouble controlling *which* links to deactivate.

Independent of its effect on modularity or fitness, the tactic of reducing HyperNEAT links by increasing the ZeroOutHalfWidth value did have the desired effect of lowering the number of ANN links. This technique may be beneficial to future HyperNEAT users that wish to reduce the number of links in HyperNEAT-generated ANNs.

4.4 Retina with Increased Geometric Coordinate Separation

Another technique that could facilitate the production of phenotypic modules in HyperNEAT's ANNs would be to make it easier for HyperNEAT's CPPNs to discriminate between the nodes on the left and right sides of the ANN. This goal can be accomplished by changing the geometric representation of the problem, which means changing the Cartesian coordinates assigned to different nodes in the ANN. Because each CPPN computes the weights of links between nodes as a function of the geometric locations of those nodes, changing these coordinate values can bias HyperNEAT toward different types of phenotypes that perform significantly differently [5]. Moreover, the intuitions human engineers have for how to geometrically represent problems can also aid the performance of HyperNEAT [5].

This method is not guaranteed to work, however, because there are ways to create modularity that do not respect the left and right sides of the coordinate space, and this mechanism might bias the CPPN away from producing them. Nevertheless, this technique of spreading the nodes out in coordinate space could at least make it easier to adopt the left-right modularity produced by the Imposed Modularity treatment. Such left-right modularity is also likely to be the type a human engineer would apply to this problem.

We implemented this method by changing the coordinate values of the nodes from a representation that had already been designed to encourage left-right modularity (Figure 2b) to one that separated the left and right nodes in geometric space even further (Figure 2c). This geometric separation did not increase performance in any treatment compared to the Standard Setup (compare Figure 3a to 3c). Performance actually decreased slightly for the Left & Right problem, and decreased noticeably for the MVG treatments ($p < 0.05$, Mann-Whitney U rank test). Although the effect was not dramatic, this result confirms a previous finding that different

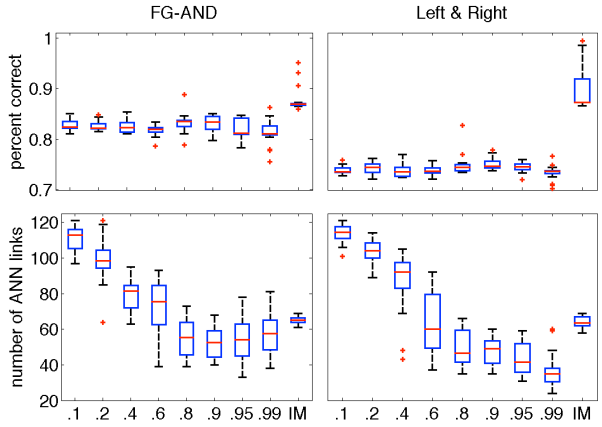

Figure 5. The effect on performance (top row) and the number of ANN links (bottom row) of varying the ZeroOutHalfWidth parameter. Each column represents a treatment with a different ZeroOutHalfWidth value (columns 1-8) or the Imposed Modularity (IM) treatment (column 9), which is shown for comparison. The midline shows the mean, the lower and upper box lines show the 25th and 75th percentiles, the whiskers enclose all non-outliers, and outliers are shown as asterisks.

geometric representations of a problem can affect HyperNEAT's performance [5]. Computational limits prevented us from testing additional geometric representations, but it is not obvious how to create a geometric layout that would substantially increase the left-right bias more than the representation we tested. Based on these tests, we believe it is unlikely that changes to the geometric representation alone will make a substantial improvement in HyperNEAT's performance on the Retina Problem.

4.5 Simplified Retina Problem

The previous experiments in this paper have failed to demonstrate that HyperNEAT is capable of producing modular ANNs. Instead, its ANNs tend to be more fully connected than modular ANNs would be. One explanation for these results is that HyperNEAT is simply incapable of generating modular ANNs. To test this hypothesis, we conducted experiments on a simple task that explicitly requires modularity. In this *Simplified Retina Problem*, there are eight inputs and two outputs (Figure 6). The goal of the network is to output the sum of the left four inputs in the left output, and the sum of the right four inputs in the right output. The output nodes had linear activation functions instead of sigmoid functions. The correct wiring for this task is to eliminate all connections between the left inputs and the right output, and vice versa, creating two distinct modules. HyperNEAT was queried for all possible links between inputs and outputs, so it had to learn across evolutionary time to eliminate connections between the left and right sides. The fitness function rewarded networks that had smaller errors between the actual and outputted sum for the left and right sides.

Within 500 generations, all 20 runs had achieved near-perfect fitness scores (>98% of the maximum fitness). Additionally, in all but one run, the final champion had perfectly discovered the modularity of the problem by eliminating all links between the left and right sides. The sole run with imperfect modularity, which had the lowest fitness, had only one incorrect connection with a small weight value. The results from the Simplified Retina Problem allow us to reject the hypothesis that HyperNEAT is incapable of

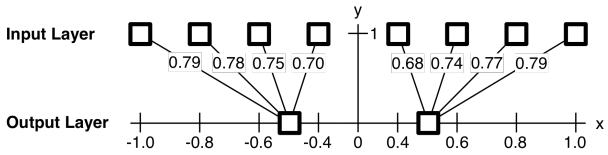

Figure 6. A modular ANN solution to the Simplified Retina Problem. Nodes (squares) are shown in their Cartesian locations. Links with a value of 0 are not shown. This champion from the end of a run has a nearly perfect fitness score because HyperNEAT created a modular ANN by deactivating links between the left and right sides.

producing modular ANNs. This experiment provides the first documented case of HyperNEAT producing modular neural networks, albeit on a very simple problem.

5. DISCUSSION & CONCLUSION

In contrast to other generative encodings that were explicitly designed to produce modularity, there is no *a priori* reason to expect HyperNEAT will produce modular networks. Nevertheless, it would be beneficial if HyperNEAT could generate modularity when doing so would improve performance. In this paper we tested whether HyperNEAT would generate modular ANNs on a suite of problems where both reason and experimental evidence suggest that modularity is helpful. HyperNEAT performed poorly on the problems and did not generate modular networks. Imposing modularity improved HyperNEAT's performance, indicating that HyperNEAT would have performed better had it been able to generate such modularity on its own. These results suggest that HyperNEAT has difficulty generating modularity on complex problems, although more research is necessary to determine if these results generalize to other problems.

Even with imposed modularity, HyperNEAT did not handle the MVG regime as effectively as Kashtan and Alon's direct encoding, where the MVG treatments outperformed the FG treatments. However, FT-NEAT, a direct encoding control for HyperNEAT, also did not generate modular and high-performing networks in MVG environments. Our control experiments thus suggest that the reason the MVG environments do not qualitatively differ from the FG environments is likely explained by differences between the experiment implementations. One candidate explanation is that Kashtan and Alon's experiments have a smaller search space. For example, the link weights and thresholds seem to be discrete values with only a few options, instead of the continuous values in our experiments. Mutations between a few discrete values may have a larger effect on the network output, making it more likely that single mutations can switch between a solution to FG-OR and FG-AND. Kashtan and Alon report that networks in the MVG regimes evolved to switch between solutions to FG-OR and FG-AND with a single mutation [13]. If such a switch requires multiple mutations in our implementation, or a single rare mutation, evolution may be unlikely to benefit from modular phenotypes because it cannot quickly rearrange modules. We will investigate this hypothesis in future work. However, even without the benefits of reorganization (e.g., in the unchanging Retina Left & Right treatment), imposed modularity benefitted HyperNEAT, so we cannot conclude that HyperNEAT had no incentive to produce modularity.

Despite HyperNEAT's difficulties with generating modularity on variants of the Retina Problem, we showed that it is capable of producing modular ANNs on the Simplified Retina Problem. While these results demonstrate for the first time that HyperNEAT can

generate modular phenotypes, the results from variants of the more complicated Retina Problem suggest that more research is needed to understand how to generate modular ANNs via HyperNEAT on complex problems.

Given the promise of the HyperNEAT approach to evolving complex ANNs, it is worthwhile to investigate the degree to which it produces phenotypic regularity, modularity, and hierarchy, which are traits that facilitate the evolution of complexity in natural organisms. While HyperNEAT excels at producing regular phenotypes [3, 4, 7], it was unknown whether it produced modular and hierarchical phenotypes. This paper demonstrates that HyperNEAT can generate modular phenotypes on a simple problem, but our results suggest that it may struggle to do so on more complex problems. In our future work we will investigate how to increase HyperNEAT's ability to evolve modular phenotypes on complex problems. We will also study how HyperNEAT might be able to evolve artificial neural networks that are hierarchical.

6. ACKNOWLEDGMENTS

We thank Kenneth O. Stanley, Robert T. Pennock, and the anonymous reviewers for their helpful comments. This work was supported in part by NSF grants CCF-0643952, CCF-0750787, CNS-0751155, CCF-0820220, CCF-0523449, and CNS-0915885, U.S. Army Grant W911NF-08-1-0495, a Quality Fund Grant from MSU, and the DARPA FunBio program.

7. REFERENCES

[1] S. B. Carroll, Endless forms most beautiful: The new science of evo devo and the making of the animal kingdom. (New York): W. W. Norton & Company, 2005.

[2] J. Clune, B. E. Beckmann, R. T. Pennock, C. Ofria, "HybrID: A Hybridization of Indirect and Direct Encodings for Evolutionary Computation." Proceedings of the European Conference on Artificial Life, 2009.

[3] J. Clune, B. E. Beckmann, C. Ofria, and R. T. Pennock, "Evolving coordinated quadruped gaits with the HyperNEAT generative encoding." Proceedings of the IEEE Congress on Evolutionary Computing, pp. 2764-2771, 2009.

[4] J. Clune, C. Ofria, and R. T. Pennock, "How a generative encoding fares as problem-regularity decreases." Parallel Problem Solving from Nature, pp. 358–367, 2008.

[5] J. Clune, C. Ofria, R. T. Pennock, "The sensitivity of Hyperneat to different geometric representations of a problem." Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 675-682, 2009.

[6] D. B. D'Ambrosio and K. O. Stanley, "Generative encoding for multiagent learning." Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 819–826, 2008.

[7] J. Gauci and K. O. Stanley, "A case study on the critical role of geometric regularity in machine learning." AAAI Conference on Artificial Intelligence, pp. 628–633, 2008.

[8] F. Gruau, "Automatic definition of modular neural networks." Adaptive Behaviour, 3(2): 151–183, 1995.

[9] L. H. Hartwell, J. H. Hopfield, S. Leibler, and A. W. Murray, "From molecular to modular cell biology." Nature, 402: C47-C52, 1999.

[10] G. S. Hornby, "Measuring, enabling and comparing modularity, regularity and hierarchy in evolutionary design." Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 1729-1736, 2005.

[11] G. S. Hornby, J. B. Pollack, "Creating high-level components with a generative representation for body-brain evolution." Artificial Life, 8(3), 2002.

[12] G. S. Hornby, H. Lipson, and J. B. Pollack, "Generative representations for the automated design of modular physical robots." IEEE Transactions on Robotics and Automation, 19: 703–719, 2003.

[13] N. Kashtan and U. Alon, "Spontaneous evolution of modularity and network motifs." Proceedings of the National Academy of Sciences, 102: 13773-13779, 2005.

[14] C. P. Klingenberg, Developmental constraints, modules and evolvability. In Variation: a Central Concept in Biology (Hallgrimsson B and Hall BK, eds), p. 219-247, Elsevier, 2005.

[15] A. Kreimer, E. Borenstein, U. Gophna, E. and Ruppin, "The evolution of modularity in bacterial metabolic networks." Proceedings of the National Academy of Sciences, 105: 6976-6981, 2008.

[16] A. Lindenmayer, "Mathematical models for cellular interaction in development, parts I and II." J. Theoretical Biol., 18: 280–299, 1968.

[17] H. Lipson, "Principles of modularity, regularity, and hierarchy for salable systems." Journal of Biological Physics and Chemistry, 7: 125-128, 2007.

[18] M. Parter, N. Kashtan, and U. Alon, "Environmental variability and modularity of bacterial metabolic networks." BMC Evolutionary Biology, 7:169 –195, 2007.

[19] M. Parter, N. Kashtan, and U. Alon, "Facilitated variation: How evolution learns from past environments to generalize to new environments." PLoS Computational Biology, 4: e1000206, 2008.

[20] R.A. Raff, The Shape of Life: Genes, Development, and the Evolution of Animal Form. The University of Chicago Press, 1996.

[21] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies." Evolutionary Computation, 10(2): 99–127, 2002.

[22] K. O. Stanley and R. Miikkulainen, "A Taxonomy for Artificial Embryogeny." Artificial Life, 9: 93-130, 2003.

[23] K. O. Stanley, "Compositional pattern producing networks: A novel abstraction of development." Genetic Programming and Evolvable Machines, 8: 131-162, 2007.

[24] K. O. Stanley, D. B. D'Ambrosio and J. Gauci, "A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks." Artificial Life. 15(2): 185-212, 2009.

[25] G.P. Wagner and L. Altenberg. "Complex adaptations and the evolution of evolvability." Evolution, 50: 967-976, 1996.

[26] G. P. Wagner, "Homologues, natural kinds and the evolution of modularity." American Zoologist, 36: 36-43, 1996.

[27] www.picbreeder.org