Evolving the Placement and Density of Neurons in the HyperNEAT Substrate

In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)*. New York, NY: ACM
Winner of the Best Paper Award in the Generative and Developmental Systems (GDS) Track

Sebastian Risi
School of EECS
University of Central Florida
Orlando, FL 32816, USA
sebastian.risi@gmail.com

Joel Lehman
School of EECS
University of Central Florida
Orlando, FL 32816, USA
jlehman@eecs.ucf.edu

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

ABSTRACT

The *Hypercube-based NeuroEvolution of Augmenting Topologies* (HyperNEAT) approach demonstrated that the pattern of weights across the connectivity of an artificial neural network (ANN) can be generated as a function of its geometry, thereby allowing large ANNs to be evolved for high-dimensional problems. Yet it left to the user the question of where hidden nodes should be placed in a geometry that is potentially infinitely dense. To relieve the user from this decision, this paper introduces an extension called *evolvable-substrate HyperNEAT* (ES-HyperNEAT) that determines the placement and density of the hidden nodes based on a *quadtree*-like decomposition of the hypercube of weights and a novel insight about the *relationship* between connectivity and node placement. The idea is that the representation in HyperNEAT that encodes the pattern of connectivity across the ANN contains implicit information on where the nodes should be placed and can therefore be exploited to avoid the need to evolve explicit placement. In this paper, as a proof of concept, ES-HyperNEAT discovers working placements of hidden nodes for a simple navigation domain on its own, thereby eliminating the need to configure the HyperNEAT substrate by hand and suggesting the potential power of the new approach.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning – connectionism and neural nets

General Terms

Algorithms

Keywords

Substrate Evolution, NEAT, HyperNEAT, Neuroevolution

1. INTRODUCTION

When the Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) method was introduced [5, 9, 17], it provided a new perspective on evolving artificial neural networks (ANNs) by showing that the pattern of weights across the connectivity of an ANN can be generated as a function if its geometry. This insight allowed large ANNs with regularities in connectivity to evolve for high-dimensional problems [4, 5, 6, 9, 10, 17]. Yet the *positions* of the nodes connected through this approach must be decided *a priori* by the user. Thus by introducing the idea that connectivity is related to geometry, HyperNEAT also uncovered a deep and enduring puzzle about the placement of the nodes themselves: What kind of representation can decide *where to place nodes* in a geometry that is potentially infinitely dense? It seems that, given that there are approximately 100 billion neurons in the human brain [13], for any representation to evolve the placement and density of nodes it would need to span an incalculable gulf between networks of several dozen nodes and several billion. The contributions of this paper are to introduce a novel insight about the *relationship* between connectivity and node placement that suggests they are really two sides of the same coin, and to exploit this relationship to avoid the need to evolve the placement of nodes at all.

That is, the novel insight is that a representation that encodes the pattern of connectivity across a network (such as in HyperNEAT) automatically contains *implicit* clues on where the nodes should be placed to best capture the information stored in the connectivity pattern. In other words, there is no need for any new information or any new representational structure beyond the very same *compositional pattern producing network* (CPPN) that already encodes network connectivity in HyperNEAT.

In this paper, the *evolvable-substrate HyperNEAT* (ES-HyperNEAT) algorithm is introduced that constructs the final phenotypic network, including the number, density, and placement of nodes, guided by information implicit in the CPPN. A simple navigation domain is presented in which the agent needs to make decisions that are not linearly separable at the turning points of the maze, thereby requiring an ANN controller with hidden nodes. The task thereby serves as an initial proof of concept for the novel approach.

Because the task is simple, it can be solved effectively by HyperNEAT with a traditional fixed-geometry substrate in which the positions of a few hidden nodes are decided by the user. Yet the ES-HyperNEAT approach finds a solution in about the same number of evaluations *and* frees the user from deciding the placement and number of hidden nodes. Importantly *no new information or parameters* were added to the CPPN representation beyond that in original HyperNEAT. Thus the entire internal geometry of node placement

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

and density is *derived* from the connectivity pattern successfully. The advantage of deriving node placement from connectivity is further confirmed by an additional experiment that shows that representing evolved node placement *independently* of connectivity yields worse performance.

While the proof-of-concept experiment is simple and does not take advantage of the full power of the approach, its long-term impact should be to enable very complex tasks for which large-scale unknown node placements are needed to be solved in the future. The more complex the task, the more important it will be to free the user from the burden of configuring the substrate by hand. Also importantly, the approach has the potential to create networks from several dozen nodes up to several million, which will be necessary in the future to create truly intelligent systems. It is furthermore an important contribution to generative and developmental systems (GDS) because indirect encodings of connectivity patterns (such as in HyperNEAT) are among its most promising applications, and the insight that node placement is implicit within connectivity itself can accelerate research in this direction.

2. BACKGROUND

This section reviews NEAT and HyperNEAT, which are foundational to the approach introduced in this paper.

2.1 Neuroevolution of Augmenting Topologies

The HyperNEAT method that enables learning from geometry in this paper is an extension of the original NEAT algorithm that evolves ANNs through a *direct* encoding. NEAT performs well in a variety of control and decision-making problems [1, 18, 20]. It starts with a population of simple neural networks and then *complexifies* them over generations by adding new nodes and connections through mutation. By evolving networks in this way, the topology of the network does not need to be known a priori; NEAT searches through increasingly complex networks to find a suitable level of complexity.

The important feature of NEAT for the purpose of this paper is that it evolves *both* the topology and weights of a network. Because it starts simply and gradually adds complexity, it tends to find a solution network close to the minimal necessary size. In principle, another method for evolving or learning the topology and weights of networks could also play the role of NEAT in this paper. Nevertheless, what is important is to begin with a principled approach to learning both such features, which NEAT provides. For a complete overview of NEAT see Stanley and Miikkulainen [18, 20].

The next section reviews the HyperNEAT extension to NEAT that is itself extended in this paper.

2.2 HyperNEAT

In direct encodings like NEAT, each part of the solution's representation maps to a single piece of structure in the final solution [8]. The significant disadvantage of this approach is that even when different parts of the solution are similar, they must be encoded and therefore discovered separately. Thus this paper employs an *indirect* encoding instead, which means that the description of the solution is compressed such that information can be reused, allowing the final solution to contain more components than the description itself. Indirect encodings are powerful because they allow solutions to be represented as a *pattern* of parameters, rather than requiring each parameter to be represented individually [2, 3, 10, 12, 16, 19]. HyperNEAT, reviewed in this section, is an indirect encoding extension of NEAT that is proven in a number of challenging domains that require discovering regularities [4, 6, 9, 10, 17]. For a full description of HyperNEAT see Stanley et al. [17] and Gauci and Stanley [10].

In HyperNEAT, NEAT is altered to evolve an indirect encoding called *compositional pattern producing networks* (CPPNs [16]) *instead* of ANNs. CPPNs, which are also networks, are designed to encode *compositions of functions*, wherein each function in the composition loosely corresponds to a useful regularity. For example, a Gaussian function induces symmetry. Each such component function also creates a novel geometric *coordinate frame* within which other functions can reside. For example, any function *of* the output of a Gaussian alone will output a symmetric pattern because the Gaussian is symmetric.

The appeal of this encoding is that it allows spatial patterns to be represented as networks of simple functions (i.e. CPPNs), which means that NEAT can evolve CPPNs just like ANNs. CPPNs are similar to ANNs, but they rely on more than one activation function (each representing a common regularity) and are an abstraction of biological development (through which phenotype patterns are constructed) rather than of brains.

The indirect CPPN encoding can compactly encode patterns with regularities such as symmetry, repetition, and repetition with variation [15, 16]. For example, simply by including a Gaussian function, which is symmetric, the output pattern can become symmetric. A periodic function such as sine creates segmentation through repetition. Most importantly, *repetition with variation* (e.g. such as the fingers of the human hand) is easily discovered by combing regular coordinate frames (e.g. sine and Gaussian) with irregular ones (e.g. the asymmetric x-axis). For example, a function that takes as input the sum of a symmetric function and an asymmetric function outputs a pattern with imperfect symmetry. In this way, CPPNs produce regular patterns with subtle variations. The potential for CPPNs to represent patterns with motifs reminiscent of patterns in natural organisms has been demonstrated in several studies [15, 16].

The main idea in HyperNEAT is that CPPNs can naturally encode *connectivity patterns* [9, 10, 17]. That way, NEAT can evolve CPPNs that represent large-scale ANNs with their own symmetries and regularities.

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 *also* denotes the connection between the two-dimensional points (x_1, y_1) and (x_2, y_2), and the output of the CPPN for that input thereby represents the weight of that connection (figure 1). By querying every possible connection among a pre-chosen set of points in this manner, a CPPN can produce an ANN, wherein each queried point is a neuron position. Because the connections are produced by a function of their endpoints, the final structure is produced with *knowledge* of its geometry. In effect, the CPPN is painting a pattern on the inside of a four-dimensional hypercube that is interpreted as the isomorphic connectivity pattern, which explains the origin of the name *hypercube-based NEAT* (HyperNEAT). Connectivity patterns produced by a CPPN

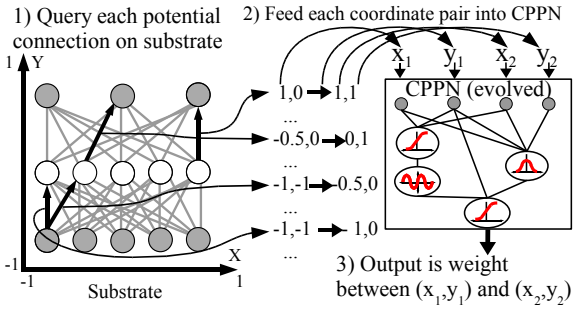


Figure 1: Hypercube-based Geometric Connectivity Pattern Interpretation. A collection nodes, called the *substrate*, is assigned coordinates that range from -1 to 1 in all dimensions. (1) Every potential connection in the substrate is queried to determine its presence and weight; the dark directed lines in the substrate depicted in the figure represent a sample of connections that are queried. (2) Internally, the CPPN (which is evolved) is a graph that determines which activation functions are connected. As in an ANN, the connections are weighted such that the output of a function is multiplied by the weight of its outgoing connection. For each query, the CPPN takes as input the positions of the two endpoints and (3) outputs the weight of the connection between them. Thus, CPPNs can produce regular patterns of connections in space.

in this way are called *substrates* so that they can be verbally distinguished from the CPPN itself, which has its own internal topology.

Each queried point in the substrate is a node in an ANN. In current implementations of HyperNEAT as of this writing, the experimenter defines both the location and role (i.e. hidden, input, or output) of each such node. As a rule of thumb, nodes are placed on the substrate to reflect the geometry of the task [4, 6, 9, 17]. That way, the connectivity of the substrate is a function of the task structure.

For example, the sensors of an autonomous robot can be placed from left to right on the substrate in the same order that they exist on the robot. Outputs for moving left or right can also be placed in the same order, allowing HyperNEAT to understand from the outset the correlation of sensors to effectors. In this way, knowledge about the problem geometry can be injected into the search and HyperNEAT can exploit the regularities (e.g. adjacency, or symmetry) of a problem that are invisible to traditional encodings. Yet a problem that has endured with HyperNEAT is that the experimenter is left to decide how many hidden nodes there should be and where to place them. That is, although the CPPN determines how to connect nodes in a geometric space, it does not specify where the nodes should be. No automated solution to this problem has yet been introduced, in part because it seems to require some kind of additional structure or apparatus beyond the CPPN representation.

In answer to this challenge, the next section introduces an extension to HyperNEAT in which the placement and density of the hidden nodes do not need to be set a priori and in fact are completely determined by implicit information in the CPPN itself.

3. CHOOSING CONNECTIONS TO EXPRESS

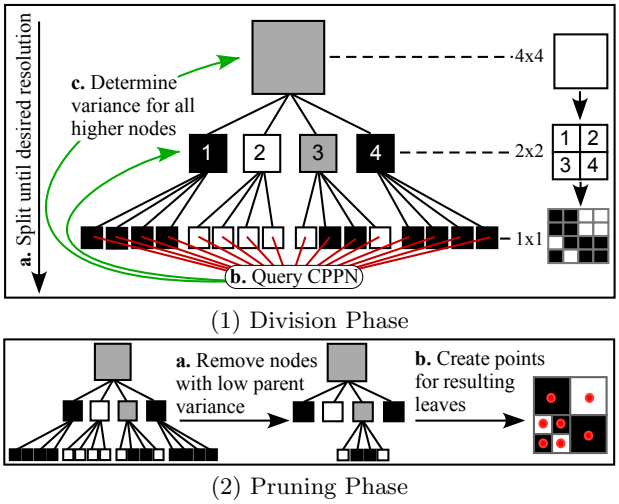
The placement of nodes in original HyperNEAT is decided by the user. Yet whereas it is often possible to determine how sensors and effectors relate to domain geometry, it is difficult for the user to determine the best placement and number of necessary hidden nodes *a priori*. For example, the location of the hidden nodes in the substrate in figure 1 had to be decided by the user. HyperNEAT thus creates the strange situation that it can decide with what weight any two nodes in space should be connected, but it cannot tell us anything about where the nodes should be. What representation can evolve the placement and density of nodes that can potentially span between networks of several dozen nodes and several billion?

3.1 Implicit Information in the Hypercube

The novel insight in this paper is that a representation that encodes the pattern of connectivity across a network automatically contains implicit information on where the nodes should be placed. In HyperNEAT the pattern of connectivity is described by the CPPN, where every point in the four-dimensional space denotes a potential connection between two two-dimensional points. Because the CPPN takes x_1, y_1, x_2, and y_2 as input, it is a function of the *infinite* continuum of possible coordinates for these points. In other words, the CPPN encodes a potentially infinite number of connection weights within the hypercube of weights. Thus one interesting way to think about the hypercube is as a theoretically infinite pattern of possible connections that *might* be incorporated into a neural network substrate. If a connection is chosen to be included, then by necessity the nodes that it connects must *also* be included in the substrate. Thus by asking which connections to include from the infinite set, we are also asking which nodes to include.

By shifting the question of what to include in the substrate from nodes to connections, two important insights follow: First, the more such connections are included, the more nodes would also be added to the substrate. Thus the *node density* increases with the number of connections. Second, for any given pattern, there is some density above which increasing density further offers no advantage. For example, if the hypercube is a uniform gradient of maximal connection weights (i.e. all weights are the same constant), then in effect it encodes a substrate that computes the same function at every node. Thus adding more such nodes adds no new *information*. On the other hand, if there is a stripe of differing weights running through the hypercube, but otherwise uniform maximal connections everywhere else, then that stripe contains information that would perform a *different* function from its redundantly-uniform neighbors.

The key insight is that it is not always a good idea to add more connections because for any given finite pattern, at some resolution there is no more *information* and adding more weights at such high resolution would be redundant and unnecessary. This maximal useful resolution varies for different regions of the hypercube depending on the complexity of the pattern in those regions. Thus the answer to the question of which connections should be included is that connections should be included at high enough resolution to capture the detail (i.e. information) in the hypercube. Any more than that would be redundant. Therefore, an algorithm is needed that can choose many points to express in regions of high variance and fewer points to express in regions of relative homogeneity. Each such point is a connection weight in the substrate whose respective nodes will be expressed as well. The main principle is simple: *Density*



(1) Division Phase

(2) Pruning Phase

Figure 2: Quadtree information extraction example for a two-dimensional CPPN. The algorithm works in two main stages. (1) In the *division phase* the quadtree is created by recursively splitting each square into four new squares until the desired resolution is reached (1a). Subsequently the CPPN value for each leaf (1b) and the variance of each higher node is determined (1c). Gray nodes in the figure have a variance greater than zero. Then, in the *pruning phase* (2), all quadtree nodes are removed whose parents have a variance that is smaller than a given threshold (2a). Points are created for all resulting quadtree leaves (2b). That way, the density of points in different regions will correspond to the amount of *information* in that region.

follows information. Thus a deep insight in this argument is that the placement of nodes in an ANN is ultimately a signification of where information is stored within weights.

To perform the task of choosing points (i.e. weights) to express, a data structure is needed that allows space to be represented at variable levels of granularity. One such multi-resolution technique is the *quadtree* [7], which traditionally describes two-dimensional regions. It has been applied successfully in fields ranging from pattern recognition to image encoding [14, 21] and is based on recursively splitting a two-dimensional region into four sub-regions. That way, the decomposition of a region into four new regions can be represented as a subtree whose parent is the original region with one descendent for each decomposed region. The recursive splitting of regions can be repeated until the desired resolution is reached or no further subdivision is needed.

How can this data structure be applied to choose which points in the hypercube to express? To facilitate understanding without loss of generality, the basic idea behind the algorithm is explicated in two dimensions (i.e. the CPPN takes two inputs). Of course, in practice it is run in four dimensions for a four-dimensional CPPN.

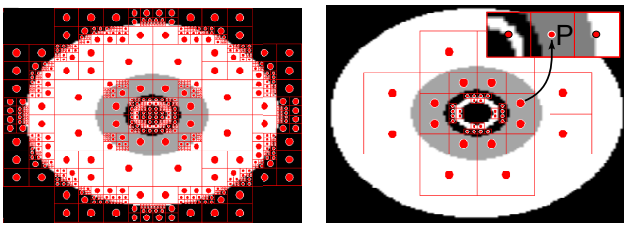
The quadtree algorithm works in two main phases (figure 2): In the **division phase** the quadtree is created by recursively subdividing the initial square until a desired initial resolution r is reached (e.g. 4×4). Once this resolution is reached, for every *quadtree leaf* (corresponding to square (x_1, y_1, x_2, y_2)), the CPPN is queried at position $\left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right)$ and the resulting value w is stored. Given those values $(w_1, w_2, .., w_k)$ for a subtree of quadtree node p and mean $\bar{w}$ the variance of node p in the quadtree can be calculated as $\sigma_p^2 = \frac{1}{k}\sum_1^k(\bar{w}-w_i)^2$. This variance is a heuristic indicator of the homogeneity (i.e. lack of information) of a region. If the variance of the parent of a quadtree leaf is still higher than a given division threshold d_t then the division phase can be reapplied for the corresponding square, allowing increasingly high densities. A maximum resolution level r_m can be set to place an upper bound on the number of possible nodes.

The quadtree representation created in the division phase can serve as a heuristic variance indicator to decide on the placement and density of points to express. Because more points should be expressed the higher the variance is in a certain region, a **pruning phase** is executed, in which quadtree nodes are removed whose parents' variance is smaller than a given variance threshold σ_t^2. Subsequently, points are created for all resulting quadtree leaf nodes. The result is higher resolution in areas of more variation.

Figure 3a shows an example of the points chosen at this stage of the algorithm, which resembles typical quadtree image decompositions [21]. The variance is high at the borders of the circles, which results in a high density of expressed points at those locations. However, there are reasons to refine the quadtree algorithm further to best suit the purpose of choosing connections for expression in a substrate. One way to think about the pattern expressed by the CPPN (e.g. such as the one in figure 3) is as a kind of *language* through which the CPPN describes where it wants to express connections. Like any language, it can be *interpreted* in different ways. The way it is interpreted will bias the ES-HyperNEAT algorithm towards different kinds of node density patterns. The goal should be to provide the CPPN with a most convenient language for expressing points in locations that would be useful for describing the weights of an ANN. While figure 3a takes a step in this direction, it tends to cluster many points along *edges* (because they represent areas of high variance). Yet clustering density around edges would make it hard for the CPPN to easily choose to express weights of one value and not another because the points chosen along edges are on both sides. Thus a more parsimonious "language" for describing density patterns would ignore the edges and focus on the inner region of *bands*, which are points that are enclosed by at least two neighbors on opposite sides (e.g. left and right) with different CPPN activation levels (figure 3b). Furthermore, narrower bands can be interpreted as requests for more point density, giving the CPPN an explicit mechanism for affecting density.

Thus, to facilitate banding, a second pruning stage is added that removes points that are not in a band. Membership in a band for square (x_1, y_1, x_2, y_2) is determined by $b = \max(\min(d_{\text{top}}, d_{\text{bottom}}), \min(d_{\text{left}}, d_{\text{right}}))$, where d_{left} is the difference in CPPN activation levels between the point $\left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right)$ and its left neighbor at the same resolution level with location $\left(\frac{2x_1-|x_1-x_2|}{2}, \frac{y_1+y_2}{2}\right)$ (the other values, d_{right}, d_{bottom}, and d_{top}, are calculated accordingly). If the band level b is below a given threshold b_t then the corresponding point is not expressed. Figure 3b shows the resulting point selections with band pruning.

This approach also naturally enables the CPPN to increase the density of points chosen by creating more bands or making them thinner. Thus no new information and no new representational structure beyond the CPPN already employed in HyperNEAT is needed to encode network connectivity, as concluded in the next section.



(a) Quadtree point selection without band pruning (b) Final point selection with band pruning

Figure 3: Example point selection in two dimensions. Different stages of point selection are shown for a two-dimensional image. Chosen points are shown in (a) after the pruning stage but without band pruning. Points that still remain after band pruning (e.g. point P, whose neighbors at the same resolution have different CPPN activation levels) are shown in (b). Note that the points in the outer black region are removed because they are not enclosed by contrasting neighbors on opposite sides. The resulting point distribution reflects the information inherent in the image. In effect, the CPPN speaks a *language* through its pattern that denotes exactly which weights within the hypercube should be expressed.

Input: 4-D CPPN, input & output nodes on substrate
Output: Final ANN
1) Division Phase
a. Create hypertree by recursively dividing hypercube until desired resolution r is reached.
b. Query CPPN for each leaf for weight w.
c. Determine variance for all tree nodes.
d. Repeat division on leaves with parent variance $\sigma_p^2 > d_t$ (and maximum resolution r_t not reached).
2) Pruning Phase
a. Remove all nodes with parent variance $\sigma_p^2 < \sigma_t^2$.
b. For all leaf nodes: If $|w|>w_t$ and band level $b > b_t$ create connection with a weight proportional to w and corresponding hidden neurons if not existent.
3) Integration Phase
a. Query connections to outputs from hidden nodes without any *outgoing* connections.
b. Query connections from inputs to output and hidden nodes without any *incoming* connections.
c. For (a) and (b): If $|w| > w_t$ create connection with a weight proportional to w.

Algorithm 1: Evolvable-Substrate HyperNEAT

3.2 Evolvable-Substrate HyperNEAT (ES-HyperNEAT)

The weight-choosing approach from the previous section can be generalized to work in the four-dimensional hypercube that is isomorphic to a two-dimensional connectivity pattern. The quadtree is replaced by what can be called a *hypertree*, in which each node in the tree has 16 descendents. The band pruning in the hypercube takes four dimensions into account instead of only two as in the previous section. Recall that the *weights* of connections are being chosen in the hypercube, whose nodes (i.e. neurons) will be placed in the substrate with them. In particular, for each chosen connection the corresponding hidden neurons are included.

A third step is also added to the algorithm, called the **integration phase**, that constructs the final fully-functioning ANN from the discovered hidden nodes by connecting them to the inputs and outputs. Integration entails first querying potential connections from hidden nodes without *outgoing* connections to outputs and then sequentially querying possible connections to output and hidden nodes that remain without *incoming* connections from input neurons.

The complete ES-HyperNEAT algorithm that determines the density and locations of the hidden nodes is enumerated in Algorithm 1. The six parameters, weight threshold w_t, band threshold b_t, variance threshold σ_t^2, division threshold d_t, initial resolution r, maximum resolution r_m, and the space spanned by the hypercube, adjust the amount and distribution of resulting hidden neurons and connections.

The next section demonstrates the capabilities of the substrate evolution algorithm.

4. EXPERIMENT

This section introduces the navigation task designed to demonstrate the substrate evolution algorithm.

4.1 Maze Navigation Domain

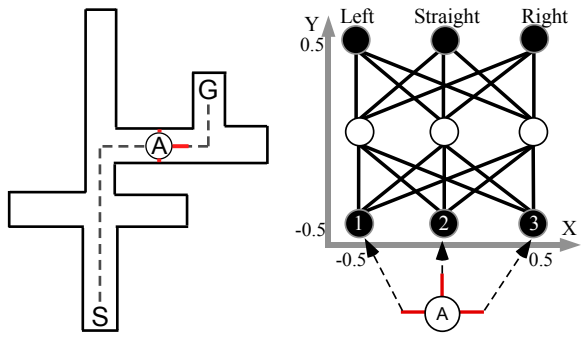
Because the goal of the substrate evolution algorithm is to determine the placement and density of the hidden nodes, a good test domain should require hidden nodes to solve. The domain in this paper is a simple navigation task that satisfies this requirement. The goal of the agent is to navigate from starting point S to goal G (figure 4a) as fast as possible. To solve the maze the agent must learn to maintain straight motion in corridors and at any junction that has an opening to the left *and* right. If there is only one opening to the left (or right) the agent should take the left (or right) branch. That way, the decision in which direction to turn to solve the given maze is similar to XOR, which is not linearly separable. It therefore requires hidden nodes and consequently makes a good validation domain for ES-HyperNEAT.

Four HyperNEAT variants are compared. In all four, the input and output nodes are placed at the same locations on the substrate (figure 4b), which are designed to geometrically correlate senses and outputs (e.g. seeing something on the left and turning left). Thus the CPPN can exploit the geometry of the agent. The agent is equipped with three rangefinder sensors that detect walls to the left, front, and right of the robot. All sensor values are scaled into the range [0,1], where lower activation indicates closer proximity to a wall. The three output neurons are *Left*, *Straight* and *Right*. At each simulated time step, the agent continues to move straight at a constant pace if the *Straight* output is greater than 0.5. If the *Straight* output is less than or equal to 0.5, the agent turns 90 degrees in the direction of the highest activated neuron (*Left* or *Right*).

The **FS-HyperNEAT** variant is the original HyperNEAT approach with a fixed substrate. To generate such a controller, a four-dimensional CPPN with inputs x_1, y_1, x_2, and y_2 queries the substrate shown in figure 4b, which has three inputs, three hidden nodes, and three output nodes, to determine its connection weights.

The **Perceptron-HyperNEAT** variant validates that the navigation task indeed needs hidden neurons. The inputs are connected directly to the outputs.

In the **ES-HyperNEAT** approach, the placement and density of the hidden nodes and their connections to the



(a) Navigation Domain (b) Fixed-Substrate Layout

Figure 4: Navigation domain and substrate configuration. (a) The goal of agent A is to navigate from starting point S to goal point G. The difficulty is that the turning decisions that the agent must make are not linearly separable and thus require an ANN with hidden nodes. The controller substrate is shown in (b) that is queried by the CPPN for FS-HyperNEAT. The autonomous agent A is equipped with three distance sensors. Three hidden nodes (shown) are given to the fixed substrate, whereas the evolvable substrate decides on the positions of hidden nodes on its own.

input and output nodes are determined entirely from the CPPN by the algorithm in Section 3.2 (Algorithm 1).

Finally, a key insight in ES-HyperNEAT is that the best clue to where to place nodes is in fact connection weights. To validate that tying node placement to connection weights indeed provides an advantage, the **SES-HyperNEAT** approach *separates* the encoding of node placement and connectivity. It is identical to ES-HyperNEAT except that it is based on *two* CPPNs, where node placement is decided by the first CPPN as in ES-HyperNEAT, but the weight of each included connection is supplied by the second CPPN.

The aim of this experiment is primarily to show that the ES-HyperNEAT algorithm can indeed evolve the placement and density of the hidden nodes on its own to solve the task. Of course, the FS-HyperNEAT variant has the advantage of a fixed topology sufficient for this simple domain, but the interesting issue is how ES-HyperNEAT will creatively lay out its substrate on its own, and whether it can do so.

The fitness function is the same for all approaches with a maximum evaluation period of 25 time steps:

$$f = \begin{cases} \frac{25}{T}, & \text{if the agent is able to reach the goal} \\ e^{-\frac{d}{D}}, & \text{otherwise,} \end{cases}$$

where d is the distance to the goal after the end of the evaluation period, D is the initial distance, and T is the number of steps it takes the agent to reach the goal.

4.2 Experimental Parameters

Because HyperNEAT differs from original NEAT only in its set of activation functions, it uses the same parameters [18]. All experiments were run with a modified version of the public domain SharpNEAT package [11]. The size of each population was 300 with 10% elitism. Sexual offspring (50%) did not undergo mutation. Asexual offspring (50%) had 0.94 probability of link weight mutation, 0.03 chance of link addition, and 0.02 chance of node addition. The NEAT coefficients for determining species similarity were 1.0 for nodes and connections and 0.1 for weights. The available CPPN activation functions were sigmoid, Gaussian, absolute value, and sine, all with equal probability of being added. Parameter settings are based on standard SharpNEAT defaults and prior reported settings for NEAT [18, 20]. They were found to be robust to moderate variation through preliminary experimentation. As in previous work [5, 17] all CPPNs received the length of the queried connection as an additional input. All hypercubes spanned the space between (-0.5, -0.5, -0.5, -0.5) and (0.5, 0.5, 0.5, 0.5).

ES-HyperNEAT and SES-HyperNEAT started with an initial resolution of $2 \times 2 \times 2 \times 2$ and had a maximum resolution of $8 \times 8 \times 8 \times 8$. The weight threshold was set to 0.6 and the band pruning threshold to 0.3. The variance and division threshold were set to 0.03. Hidden neurons were only allowed to connect to other neurons with a greater y-value. Thus all resulting ANNs are feedforward and the comparison between FS-HyperNEAT and ES-HyperNEAT (and SES-HyperNEAT) is fair.

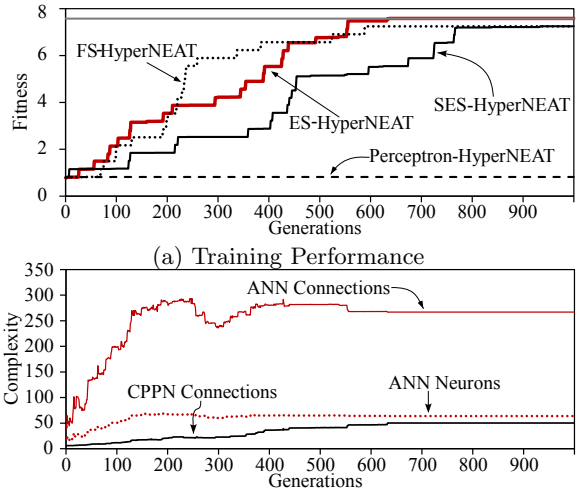
Runs consisted of 1,000 generations and all reported results are averaged over 20 runs. The problem is considered *solved* when the agent is able to navigate to the goal as fast as possible without taking any detours, which corresponds to the maximum fitness of 7.58. If no solution was found within the initial 1,000 generations, the current simulation was restarted. This procedure was repeated until a solution was found, counting all generations over all restarts for that particular run.

5. RESULTS

Figure 5a shows the training performance over generations for the fixed substrates (with three and no hidden nodes) and the evolved substrates. The substrate without hidden nodes is not able to solve the task, confirming that the domain indeed requires hidden neurons. It took FS-HyperNEAT 282 generations (σ = 139) on average to find a solution, including one restart. ES-HyperNEAT took 302 generations (σ = 189) and was able to find a solution in all 20 runs. SES-HyperNEAT took 550 generations (σ = 391) on average and required one restart. While FS-HyperNEAT solves the task slightly (though not significantly; $p = 0.76$ according to the Student's t-test) faster on average, ES-HyperNEAT reaches a slightly (though not significantly; $p = 0.33$) higher average level of fitness. Thus overall the performance of the two methods is about the same, which means that evolving the placement of the nodes exacts no extra cost in terms of evaluations in this domain. SES-HyperNEAT, however, took significantly longer to evolve solutions than ES-HyperNEAT ($p < 0.01$), indicating that it may indeed be more efficient to couple node placement and connectivity together than to encode them separately.

In ES-HyperNEAT, there is a significant positive correlation (r=0.67, p<0.01) between the number of connections in the CPPN and in the resulting ANN (figure 5b). This trend indicates that the substrate evolution algorithm may tend to create increasingly complex indirectly-encoded networks even though it is not explicitly designed to do so (e.g. like regular NEAT). The complexity of ANN (substrate) solutions (267 connections on average) is more than five times greater than that of the underlying CPPNs (50 connections on average), which suggests that ES-HyperNEAT can encode large ANNs from compact CPPN representations.

An example ANN created by ES-HyperNEAT that is able to solve the navigation domain and the CPPN that encodes



(a) Training Performance

(b) Average ES-HyperNEAT Champion Complexity

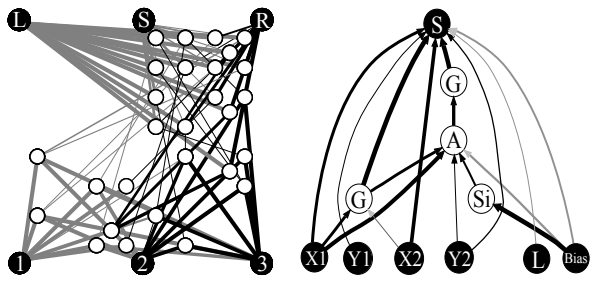
Figure 5: Training performance and average champion complexity over generations. (a) The change in performance over evaluations for HyperNEAT for the four compared approaches is shown. All results are averaged over 20 runs. FS-HyperNEAT and ES-HyperNEAT take about the same number of evaluations and the task cannot be solved without hidden neurons. SES-HyperNEAT takes significantly longer to evolve a solution. The horizontal line (top) indicates at what fitness the maze is solved. The average number of connections and neurons of the champion ANNs produced by ES-HyperNEAT and the number of connections of the underlying CPPNs are tracked in (b). Increasing CPPN complexity shows a positive (and significant) correlation with an increase in ANN complexity.

it are shown in figure 6. The algorithm creates a working network topology with hidden nodes, although it includes more structure than necessary, suggesting the technique may be most suited to finding ANNs for more complex problems.

6. DISCUSSION AND FUTURE WORK

The very idea revealed by HyperNEAT that connectivity can exist at infinite resolution hints at a potentially useful shift in thinking about networks. When continuous functions encode connectivity patterns, the strange corollary is that *nodes* are not explicitly determined; for any such connectivity pattern, the nodes that are connected can literally exist at any density and in any distribution. Yet while this condition seems to present a puzzle, it also presents an opportunity to think anew about the deeper relationship between connections and nodes. The ideas in this paper are a step towards such renewed thinking.

The convention in HyperNEAT of the last several years that the user would simply decide a priori where the nodes belong evaded this deeper mystery about how connectivity relates to node placement. In fact, if the *user* dictates that hidden node n must exist at position (a, b), it creates the unintentional constraint that any pattern of weights encoded by the CPPN must intersect position (a, b) precisely with the correct weights. That is, the pattern in the hypercube must perfectly align the correct weights through (a, b, x_2, y_2) and (x_1, y_1, a, b). Yet why should such an arbitrary *a priori* constraint on the *locations* of weights be imposed? It might be easier for the CPPN to represent the correct pattern at a



(a) ANN (b) CPPN

Figure 6: Example solution ANN created by ES-HyperNEAT and its underlying CPPN. Inputs (bottom) and outputs (top) are displayed in black. Hidden nodes are shown in white. Positive connections are dark whereas negative connections are light. Line width corresponds to connection strength. The algorithm discovered an ANN solution with hidden nodes (a) by extracting the information inherent in the simpler underlying CPPN (b). CPPN activation functions are denoted by G for *Gaussian*, S for *sigmoid*, Si for *sine*, and A for *absolute value*.

slightly *different* location in the hypercube, yet that would fail under the user-imposed convention.

One way to interpret the preceding argument is that the locations of useful information in the hypercube are where the nodes need to be. That way, the *size* of the brain is roughly correlated to its *complexity*. There is no *need* for a billion neurons to express a simple Braitenberg vehicle. Even if a billion neurons were summoned for the task, many of their functions would end up redundant, which geometrically means that large cross-sections of the hypercube would be uniform, containing no useful information. The ES-HyperNEAT approach in this paper is a heuristic attempt to formalize this notion and thereby correlate size to complexity. In this context, nodes become a kind of harbinger of complexity, proliferating where it is present and receding where it is not. Thus the solution to the mystery of the relationship between nodes and connections is that nodes are sentinels of complex connectivity; they are beacons of information in an infinite cloud of possible connections.

The contribution of this paper, supported by its results, is to show that this idea can work. While the experiment does not establish the necessity of the approach, it does establish its feasibility. It really is possible to extract from a CPPN that encodes nothing but connectivity a reasonable set of node locations based on a heuristic concept of what kind of information is important (which will be inevitably refined and improved in the future). In fact, allowing the *separate* specification of connectivity and node placement (as in SES-HyperNEAT) significantly degrades performance, providing empirical evidence for the motivation behind ES-HyperNEAT. While the navigation task is simple, it is a first step towards establishing that this theory can actually work. Future experiments in more ambitious domains will be important towards establishing the necessity of the approach.

The hypertree decomposition in this paper is likely not the last word on this type of algorithm but more fundamentally is the first of potentially many such algorithms that heuristically search the hypercube for the most promising weights. Thus in effect it opens a new research direction in neuroevolution. For example, while discovering regions of

high variance in the hypercube space can become expensive at high resolutions, a possible future refinement of the approach might begin searching only around the input neurons and then iterate to other areas of the hypercube only if they are connected.

Importantly, ES-HyperNEAT and FS-HyperNEAT take about the same number of evaluations to find a solution. Thus the new algorithm frees the user from needing to decide anything about the placement or number of hidden nodes. The ES-HyperNEAT solutions, with on average 64 hidden nodes and 267 connections, are significantly more complex than necessary for this task. Yet they are optimized as fast as a fixed substrate with only three hidden neurons. An indirectly-encoded neuroevolution algorithm is not the same as a direct encoding like NEAT that complexifies by adding one node at a time to find just the right number of nodes for a task. The promise of the indirect encoding is rather to evolve *very large* networks that would be prohibitive to such direct encodings, with thousands or more nodes. In this context, 67 nodes is still not many.

Thus, now that the technical viability of the approach is established, it is possible to move on to tasks that might benefit from large-scale ANNs, like robots driven by raw high-resolution vision, strategic game-players, or human assistants. ES-HyperNEAT offers a new angle from which to tackle such problems; for any such task, the information needed to solve it may be found lurking in the infinite depths of a hypercube of connection weights.

7. CONCLUSIONS

This paper presented a novel approach to determining the placement and density of the hidden nodes based on implicit information in an infinite-resolution pattern of weights. Results in a simple navigation task demonstrated that there is no need for any new information or any additional representational structure beyond the very same CPPN that encodes connectivity in the original HyperNEAT. The main conclusion is that ES-HyperNEAT is a promising new approach that frees the user from deciding on the placement or number of hidden nodes a priori.

Acknowledgments

This research was supported by DARPA under grant HR0011-09-1-0045 (Computer Science Study Group Phases II).

8. REFERENCES

[1] T. Aaltonen et al. Measurement of the top quark mass with dilepton events selected using neuroevolution at CDF. *Physical Review Letters*, 2009.

[2] P. J. Bentley and S. Kumar. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *Proceedings of the Genetic and Evol. Comp. Conf. (GECCO-1999)*, pages 35–43, San Francisco, 1999. Kaufmann.

[3] J. C. Bongard. Evolving modular genetic regulatory networks. In *Proceedings of the 2002 Congress on Evolutionary Computation*, 2002.

[4] J. Clune, B. E. Beckmann, C. Ofria, and R. T. Pennock. Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2009) Special Section on Evolutionary Robotics*, Piscataway, NJ, USA, 2009. IEEE Press.

[5] D. D'Ambrosio and K. O. Stanley. A novel generative encoding for exploiting neural network sensor and output geometry. In *Proc. of the Genetic and Evol. Comp. Conf. (GECCO 2007)*, NY, 2007. ACM Press.

[6] J. Drchal, J. Koutnık, and M. Šnorek. HyperNEAT controlled robots learn to drive on roads in simulated environment. In *Proc. of the IEEE Congress on Evol. Comp. (CEC 2009)*. IEEE Press, 2009.

[7] R. Finkel and J. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.

[8] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[9] J. Gauci and K. O. Stanley. A case study on the critical role of geometric regularity in machine learning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008)*, Menlo Park, CA, 2008. AAAI Press.

[10] J. Gauci and K. O. Stanley. Autonomous evolution of topographic regularities in artificial neural networks. *Neural Computation*, 2010. To appear.

[11] C. Green. SharpNEAT homepage. `http://sharpneat.sourceforge.net/`, 2003–2006.

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

[13] E. R. Kandel, J. H. Schwartz, and T. M. Jessell. *Principles of Neural Science*. McGraw-Hill, New York, fourth edition, 2000.

[14] A. Rosenfeld. Quadtrees and pyramids for pattern recognition and image processing. In *Proceedings of the 5th International Conference on Pattern Recognition*, pages 802–809. IEEE Press, 1980.

[15] J. Secretan, N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, and K. O. Stanley. Picbreeder: Evolving pictures collaboratively online. In *CHI '08: Proc. of the twenty-sixth annual SIGCHI conf. on Human factors in computing systems*, pages 1759–1768, New York, NY, USA, 2008. ACM.

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

[17] 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.

[18] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.

[19] K. O. Stanley and R. Miikkulainen. A taxonomy for artificial embryogeny. *Art. Life*, 9(2):93–130, 2003.

[20] K. O. Stanley and R. Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Art. Int. Research*, 21:63–100, 2004.

[21] P. Strobach. Quadtree-structured recursive plane decomposition coding of images. *Signal Processing*, 39:1380–1397, 1991.