

Enhancing ES-HyperNEAT to Evolve More Complex Regular Neural Networks

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

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

ABSTRACT

The recently-introduced evolvable-substrate HyperNEAT algorithm (ES-HyperNEAT) demonstrated that the placement and density of hidden nodes in an artificial neural network can be determined based on implicit information in an infinite-resolution pattern of weights, thereby avoiding the need to evolve explicit placement. However, ES-HyperNEAT is computationally expensive because it must search the entire hypercube, and was shown only to match the performance of the original HyperNEAT in a simple benchmark problem. *Iterated ES-HyperNEAT*, introduced in this paper, helps to reduce computational costs by focusing the search on a sequence of two-dimensional cross-sections of the hypercube and therefore makes possible searching the hypercube at a finer resolution. A series of experiments and an analysis of the evolved networks show for the first time that iterated ES-HyperNEAT not only matches but outperforms original HyperNEAT in more complex domains because ES-HyperNEAT can evolve networks with limited connectivity, elaborate on existing network structure, and compensate for movement of information within the hypercube.

Categories and Subject Descriptors

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

General Terms: Algorithms

Keywords: NEAT, HyperNEAT, Neuroevolution

1. INTRODUCTION

In the field of *neuroevolution*, i.e. the artificial evolution of neural networks, interest has increased in recent years in indirect network encodings, wherein the description of the solution is compressed such that information can be reused. Such compression allows the final solution to contain more components than its description. Nevertheless, neuroevolution has so far mainly produced networks with orders of magnitude fewer neurons and significantly less organization and regularity than natural brains [16, 19].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

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

Copyright 2011 ACM 978-1-4503-0557-0/11/07 ...\$10.00.

Addressing this gap, when the Hypercube-based Neuro-Evolution of Augmenting Topologies (HyperNEAT) method was introduced [6, 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 of its geometry (i.e. the position of its nodes in space). This insight allowed large ANNs with regularities in connectivity to evolve for high-dimensional problems [2, 6, 7, 17]. Yet one limitation is that the *positions* of the nodes connected through this approach must be decided *a priori* by the user.

In another step forward, *evolvable-substrate HyperNEAT* (ES-HyperNEAT), recently introduced by Risi et al. [10], showed that the placement and density of the hidden nodes can in fact be determined solely based on implicit information in an infinite-resolution pattern of weights generated by HyperNEAT. The novel insight was 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. However, ES-HyperNEAT is computationally expensive because it must search the entire hypercube, and was shown only to match the performance of the original HyperNEAT in a single benchmark [10].

Iterated ES-HyperNEAT, introduced here, remedies this cost by iteratively discovering the placement of the hidden neurons starting from the inputs and outputs of the ANN, which makes possible searching the hypercube at a finer resolution. Therefore, more ambitious experiments can be attempted, like the dual task and maze navigation domains presented in this paper. The results confirm for the first time the advantages of deriving node placement from connectivity and show that not only does iterated ES-HyperNEAT match regular HyperNEAT, but that it can in fact outperform it.

Analysis of the evolved ANNs shows that the better performance is due to ES-HyperNEAT's ability to evolve networks with limited connectivity, thereby reducing the amount of crosstalk that each neuron experiences, and its ability to elaborate on existing structure by increasing the number of connections in the ANN during evolution. Thus the approach in this paper provides a significant new practical tool for evolving large regular ANNs with less user involvement.

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 [14, 16]. 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. 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 [5]. 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. 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 [1, 7, 8, 13, 15]. 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 [2, 6, 7, 17]. For a full description of HyperNEAT see Stanley et al. [17] and Gauci and Stanley [7].

In HyperNEAT, NEAT is altered to evolve an indirect encoding called *compositional pattern producing networks* (CPPNs [13]) *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 rather than of brains.

The indirect CPPN encoding can compactly encode patterns with regularities such as symmetry, repetition, and repetition with variation [12, 13]. 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 asym-

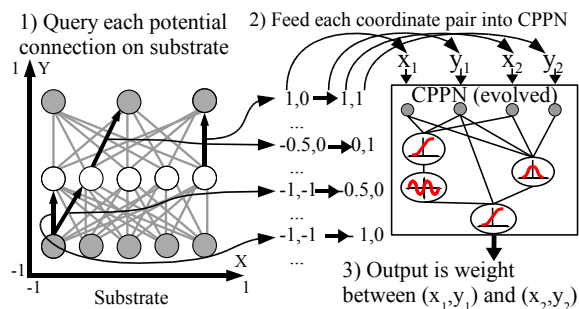


Figure 1: Hypercube-based Geometric Connectivity Pattern Interpretation. A collection nodes, called the *substrate*, is assigned coordinates that range from -1 to 1 in all dimensions. (1) Every potential connection in the substrate is queried to determine its presence and weight; the dark directed lines 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.

metric 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 [12, 13].

The main idea in HyperNEAT is that CPPNs can naturally encode *connectivity patterns* [6, 7, 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 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 the original HyperNEAT approach, 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 [2, 6, 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 with the original 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.

As a step towards allowing CPPNs also to specify the positions of the nodes and their density, Risi et al. [10] recently introduced an extension called *evolvable-substrate HyperNEAT* (ES-HyperNEAT). This approach searches for areas of high variance within the CPPN-generated pattern and places nodes there. However, ES-HyperNEAT is computationally expensive because it must search the entire hypercube, and was shown only to match the performance of the original HyperNEAT in a single benchmark [10]. The aim of this paper is to introduce a significantly more efficient version of ES-HyperNEAT (explained next) and then to show not only that it can match regular HyperNEAT, but that it can outperform it, and why.

3. ITERATED ES-HYPERNEAT

This section introduces *iterated ES-HyperNEAT*, which is an enhancement of the original ES-HyperNEAT. While the location of the hidden nodes in the substrate in figure 1 had to be decided by the user in original HyperNEAT, ES-HyperNEAT showed that the decision can in fact be automated. Yet the challenge faced by such an approach is to decide the placement and density of nodes that can potentially span between networks of several dozen nodes and several billion.

3.1 Foundational Idea: ES-HyperNEAT

The insight introduced by Risi et al. [10] is that a representation that encodes the pattern of network connectivity 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 from which some subset must be chosen to be incorporated into the ANN 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.

Another important insight is that, 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

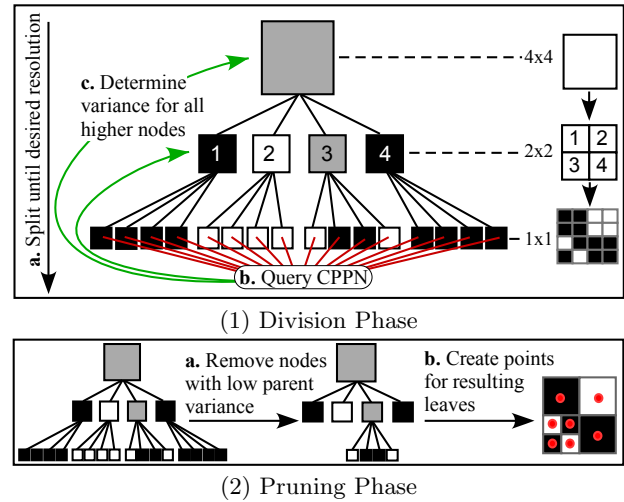


Figure 2: Quadtree information extraction example for a two-dimensional CPPN. The algorithm works in two main stages. (1) In the *division phase* the quadtree is created by recursively splitting each square into four new squares until the desired resolution is reached (1a). Subsequently the CPPN value 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.

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.

Thus the answer to the question of which connections should be included in ES-HyperNEAT 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 follows information*. In this way, 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* [4], which traditionally describes two-dimensional regions. It has been applied successfully in fields ranging from pattern recognition to image encoding [11, 18] 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 re-

ursive splitting of regions can be repeated until the desired resolution is reached or no further subdivision is needed.

The quadtree point-choosing 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 $(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2})$ and the resulting value w is stored. Given those values (w_1, w_2, \dots, 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 heterogeneity (i.e. presence 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 serves 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 region, a **pruning phase** is next executed (figure 2 bottom), in which quadtree nodes are removed whose parents’ variance is smaller than the variance threshold σ_t^2 . Subsequently, points are created for all resulting 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 [18]. The variance is high at the borders of the circles, which results in a high density of expressed points at those locations. However, Risi et al. [10] pointed out that the raw pattern output by the quadtree algorithm can be improved further. If we think of the pattern output by the CPPN as a kind of *language* for specifying the locations of expressed connections, then it makes sense to additionally to prune the points around borders so that it is easy for the CPPN to encode points safely within one region or another. 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 $(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2})$ and its left neighbor at the same resolution level with location $(\frac{2x_1 - |x_1 - x_2|}{2}, \frac{y_1+y_2}{2})$ (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

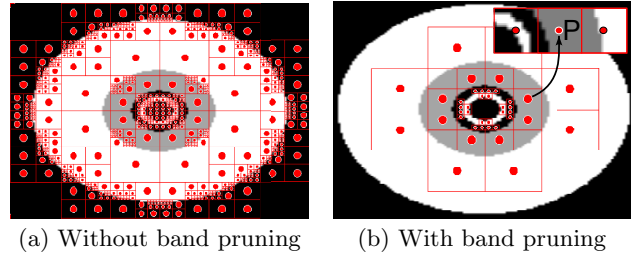


Figure 3: Example point selection in two dimensions. Chosen points are shown in (a) after the pruning stage but without band pruning. Points that still remain after band pruning (e.g. point P , whose neighbors at the same resolution have different CPPN activation levels) are shown in (b). The resulting point distribution reflects the information inherent in the image.

employed in HyperNEAT is needed to encode node placement and connectivity, as concluded in the next section.

3.2 Iterated Network Completion

The original ES-HyperNEAT approach [10] searches directly in the four-dimensional weight space, which makes discovering regions of high variance expensive at high resolutions. In fact, in four dimensions, instead of four branches for each node in the quadtree, there are 16, making such a search substantially more costly. The ES-HyperNEAT refinement presented in this section remedies this cost by iteratively discovering the placement of the hidden neurons starting from the inputs and outputs of the ANN, which turns out to make it possible to constrain the search back to two dimensions. The iterated model focuses the search within the hypercube on discovering *functional networks* in which every hidden node contributes to the ANN output and receives information (at least indirectly) from at least one input neuron, while ignoring parts of the hypercube that are disconnected. This refinement not only allows searches at higher resolutions with less computational cost but also eliminates the need for an extra integration phase of the input and output neurons, which was also part of the original ES-HyperNEAT approach.

The idea behind the algorithm is depicted in figure 4. Instead of searching directly in the four-dimensional hypercube space, the algorithm analyzes a sequence of two-dimensional cross-sections of the hypercube, one at a time, to discover which connections to include in the ANN. For example, given an input node at $(0, -1)$ the quadtree point-choosing approach is applied only to the two-dimensional *outgoing* connectivity patterns from that single node (figure 4a) described by the function $CPPN(0, -1, x, y)$ with x and y ranging from -1 to 1 . This process can be iteratively applied to the discovered hidden nodes until a user-defined maximum iteration level is reached or no more information is discovered in the hypercube (figure 4b). Similarly, starting from the output neurons the approach chooses connections based on each output’s *incoming* connectivity patterns (figure 4c). Once all hidden neurons are discovered, only those are kept that have a path to an input *and* output neuron (figure 4d).

The iterated version of ES-HyperNEAT helps to reduce computational costs by focusing the search on a sequence of two-dimensional cross-sections of the hypercube instead

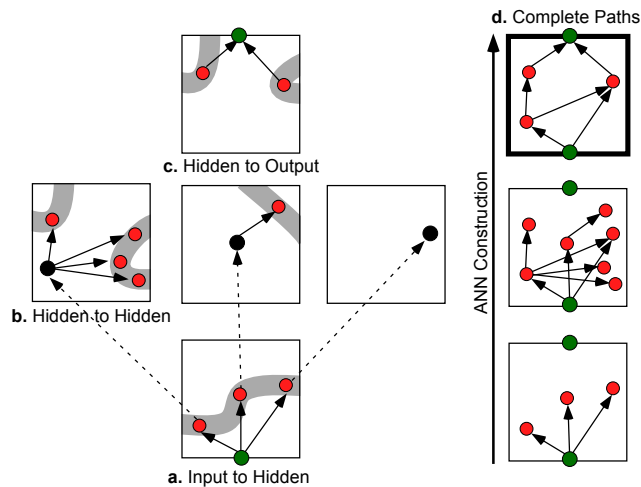


Figure 4: Iterated ES-HyperNEAT. The algorithm starts by iteratively discovering the placement of the hidden neurons from the inputs (a) and simultaneously from the outputs (c) of the ANN. The two-dimensional motif in (a) represents *outgoing* connectivity pattern from a single input node whereas the motif in (c) represents *incoming* connectivity pattern for a single output node. The target nodes discovered are those that reside within bands in the hypercube. In this refinement of ES-HyperNEAT regions of high variance are sought only in the two-dimensional cross-section of the hypercube containing the source or target node. The algorithm can be iteratively applied to the discovered hidden nodes (b). Only those nodes are kept that have a path to an input *and* output neuron (d). That way, the search through the hypercube is restricted to functional ANN topologies.

of the full four-dimensional hyperspace and therefore makes possible searching the hypercube at a finer resolution. Also, unnecessary computation is avoided. For example, if the hypercube is entirely uniform, the iterated approach will stop after the first expansion from inputs and outputs when the lack of variance in the connected cross-sections is discovered. In contrast, the original ES-HyperNEAT approach would continue to drill down n times, to its minimal necessary resolution, which costs 16^n expansions. Thus in cases such as uniform weights where such search is unnecessary, the iterated approach saves significant computation. The result is that more ambitious experiments can be attempted than in Risi et al. [10], as described next.

4. EXPERIMENTS

This section discusses two experiments designed to demonstrate new advantages of evolving the substrate.

4.1 Dual Task Domain

Organisms in nature have the ability to switch rapidly between different tasks depending on the demands of the environment. For example, a rat reacts differently when placed in a maze or in an open environment with a visible food source. The dual task domain presented here (figure 5a) will test the ability of ES-HyperNEAT and regular HyperNEAT to evolve such task differentiation.

The dual task domain consists of two *non-dependent* scenarios that require the agent to exhibit different behaviors

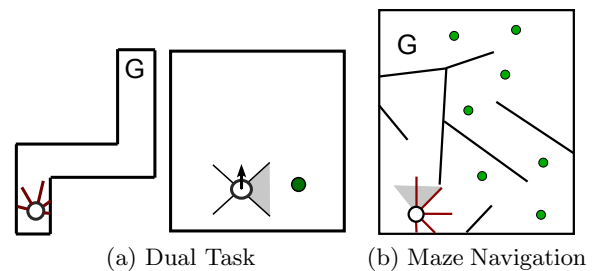


Figure 5: Domains. In the dual task domain shown in (a) the agent either has to exhibit wall-following or food-gathering behavior depending on its current environment. The goal of the agent in the maze navigation domain (b) is to reach goal point G . Because the task is deceptive the agent is rewarded for making incremental process towards the goal by following the seven waypoints.

and to react either to its rangefinders or pie-slice sensors. Because certain neurons ideally would be responsible for information that should be treated differently, this domain will likely benefit from ANNs that are not fully-connected, which the original HyperNEAT has struggled to produce in the past [3]. The hypothesis is that ES-HyperNEAT should allow the evolution of networks with limited connectivity because connections are only included at a high enough resolution to capture the information in the hypercube, thereby reducing the amount of crosstalk that each neuron experiences. In this way, ES-HyperNEAT should not only be able to match, but to outperform original HyperNEAT.

The first scenario is a simple navigation task in which the agent has to navigate from a starting point to an end point in a fixed amount of time using only its rangefinder sensors to detect walls. The fitness in this scenario is calculated as $f_{nav} = 1 - d$, where d is the distance of the robot to the goal point at the end of the evaluation scaled into the range $[0, 1]$. The second scenario is a food gathering task in which a single piece of food is placed within a square room with an agent at the center. The agent attempts to gather as much food as possible within a time limit using only its pie-slice sensors, which act as a compass towards the food item. Food only appears at one location at a time and is placed at another random location once consumed by the agent. The fitness for the food gathering task is defined by: $f_{food} = \frac{n+(1-d)}{4}$, where n corresponds to the number of collected food items (maximum four) and d is the distance of the robot to the next food item at the end of the evaluation.

The total fitness is calculated as the average performance on both scenarios. The domain is considered *solved* when the agent is able to navigate to the goal point in the first scenario and successfully collects all four food items in the second scenario, which corresponds to a fitness of 1.

4.2 Maze Navigation Domain

To evolve controllers for more complicated tasks will require a neuroevolution method that benefits from previously-discovered partial solutions to find the final solution. However, because regular HyperNEAT tends to produce fully-connected ANNs [3], it likely takes the entire set of ANN connection weights to represent a partial task solution. On the other hand, ES-HyperNEAT should be able to elaborate

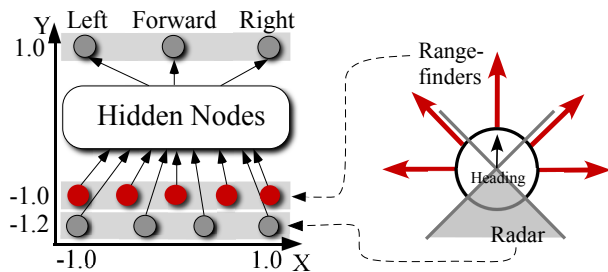


Figure 6: Substrate Configuration and Sensor Layout. The controller substrate is shown at left. Whereas the number of hidden nodes for the fixed-substrate approach is determined in advance, ES-HyperNEAT decides on the positions and density of hidden nodes on its own. The sensors layout is shown on the right. The autonomous agent is equipped with five distance and four pie-slice sensors.

on existing structure because it can increase the number of connections in the substrate during evolution.

To test this second hypothesis on when ES-HyperNEAT provides an advantage, a task is needed in which a solution is difficult to evolve directly. One such task is the deceptive maze navigation domain introduced in Lehman and Stanley [9]. In this domain (figure 5b), a robot controlled by an ANN must navigate in a maze from a starting point to an end point in a fixed time. The robot has five rangefinders that indicate the distance to the nearest wall within the maze, and four pie-slice radar sensors that fire when the goal is within the pie-slice.

If fitness is rewarded proportionally to how close the robot ends from the goal, cul-de-sacs in the maze that lead close to the goal but do not reach it are deceptive local optima. Therefore, in this paper, the fitness function f rewards the agent explicitly for discovering stepping stones towards the goal:

$$f = \begin{cases} 10, & \text{if the agent is able to reach the goal} \\ n + (1 - d), & \text{otherwise,} \end{cases}$$

where n is the number of passed waypoints (which are *not* visible to the agent) and d is the distance of the robot to the next waypoint scaled into the range $[0, 1]$ at the end of the evaluation. The idea is that agents that can reach intermediate waypoints should make good stepping stones to those that reach further waypoints.

4.3 Experimental Setup

Evolvable and fixed-substrate HyperNEAT use the same placement of input and output nodes on the substrate in both experiments (figure 6), 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 five rangefinder sensors that detect walls and four pie-slice sensors that act as a compass towards the next food item in the dual task domain or as a compass towards the goal in the maze navigation domain. All sensor values are scaled into the range $[0, 1]$, where lower activation indicates closer proximity to a wall. At each discrete moment of time, the number of units moved by the agent is $20F$, where F is the forward effector output. The agent also turns by $(L - R) * 18^\circ$, where L is the left effector output and R is the right effector output. A negative value is interpreted as a right turn.

To highlight the challenge of deciding the location and number of available hidden nodes, ES-HyperNEAT is compared to four fixed-substrate variants. **FS10x1** is the default setup with a single row of ten hidden neurons in a *horizontal* line at $y = 0$. Similar substrate layouts, like the one shown in figure 1, have shown good performance in previous research [10]. For the **FS1x10** variant ten hidden neurons are arranged *vertically* at $x = 0$. **FS5x5** has a substrate containing 25 hidden nodes arranged in a 5×5 grid. **FS8x8** tests the effects on performance of uniformly increasing the number of hidden nodes from 25 to 64 neurons. To generate such a controller for original HyperNEAT, a four-dimensional CPPN with inputs $x_1, y_1, x_2,$ and y_2 queries the substrate shown in figure 6, to determine the connection weights between the input/hidden, hidden/output, and hidden/hidden nodes. In contrast, ES-HyperNEAT decides the placement and density of nodes on its own.

4.4 Experimental Parameters

Because HyperNEAT differs from original NEAT only in its set of activation functions, it uses the same parameters [14]. All experiments were run with the HyperSharpNEAT Simulator and Experimental Platform v1.0 (available at <http://eplex.cs.ucf.edu>). 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 [14, 16]. They were found to be robust to moderate variation through preliminary experimentation. As in previous work [17] all CPPNs received the length of the queried connection as an additional input. Iterated ES-HyperNEAT had an initial and maximum resolution of 8×8 . The band pruning threshold was set to 0.3. The variance and division threshold were set to 0.03. Finally, the iteration level was 1.

5. RESULTS

All results are averaged over 20 runs. Figure 7a,b shows the training performance over generations for the HyperNEAT variants on both domains. ES-HyperNEAT solves the dual task domain in all runs and took on average 33 generations ($\sigma = 31$), whereas the best performing fixed-substrate variant, FS5x5, finds a solution in only 13 out of 20 runs. The difference in average final performance is significant ($p < 0.001$ according to the Student's t-test). These results suggest that the dual task domain benefits from ES-HyperNEAT's ability to generate networks with limited connectivity. Interestingly, the average CPPN complexity of solutions discovered by FS5x5 is 9.7 hidden nodes ($\sigma = 6.7$), almost six times higher than CPPN solutions by ES-HyperNEAT, which have 1.65 nodes on average ($\sigma = 2.2$). This difference is significant ($p < 0.05$) and indicates that dictating the location of the hidden nodes a priori makes it harder for the CPPN to represent the correct pattern.

ES-HyperNEAT also performed significantly better than the other variants in the maze navigation domain ($p < 0.001$) (figure 7b) and finds a solution in 19 out of 20 runs in 238

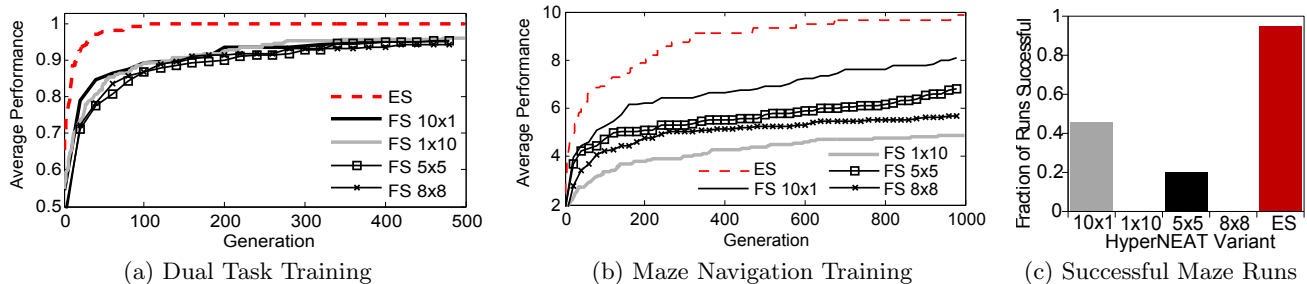


Figure 7: Average Performance. The average best fitness over generations is shown for the dual task (a) and maze navigation domain (b) for the different HyperNEAT variants, which are averaged over 20 runs. The fraction of 20 runs that successfully solve the maze navigation domain is shown in (c) for each of the HyperNEAT variants after 1,000 generations. The main result is that ES-HyperNEAT significantly outperforms all other approaches in both domains.

generations on average when successful ($\sigma = 262$). The differing performance of evolvable- and fixed-substrate HyperNEAT can also be appreciated in how frequently they solve the problem perfectly (figure 7c). ES-HyperNEAT significantly outperforms all fixed-substrate variants and finds a solution in 95% of the 20 runs. The conclusion is that evolving the substrate significantly increases performance in tasks that require incrementally building on stepping stones.

To give a sense of the computational savings from the iterated version of ES-HyperNEAT compared to the original version, every substrate from 20 iterated runs of the maze navigation task was generated by both versions. Iterated ES-HyperNEAT takes on average 0.09 seconds ($\sigma = 0.03$) and original ES-HyperNEAT takes 0.19 seconds ($\sigma = 0.06$) to generate a single substrate on the same CPU, a significant ($p < 0.001$) speedup of over two times, confirming the computational advantage of the iterated approach.

5.1 Example Solution Lineage

To gain a better understanding of how an indirect encoding like ES-HyperNEAT elaborates a solution over generations, additional evolutionary runs in the maze navigation domain were performed with sexual reproduction disabled (i.e. every CPPN has only one ancestor). This change facilitates analyzing the lineage of a single champion network. Disabling sexual reproduction did not result in a significant performance difference.

An example of four milestone ANNs in the lineage of a solution is shown in figure 8. All ANNs share common geometric features: most prominent are the symmetric network topology and denser regions of hidden neurons resembling the shape of an “H” (except the second ANN). Between generations 24 and 237 the ANN evolves from not being able to reach the first waypoint to solving the task. The solution discovered at generation 237 shows a clear holistic resemblance to generation 106 despite some general differences. Both networks have strong positive connections to the three output neurons that originate at slightly different hidden node locations. This slight shift is due to a movement of information within the hypercube for which ES-HyperNEAT can nevertheless compensate. The number of connections gradually increases from 184 in generation 24 to 356 in generation 237, indicating the incremental elaboration on existing ANN structure. Interestingly, the final ANN solves the task without feedback from its pie-slice sensors.

Figure 8 also shows that ES-HyperNEAT can encode large ANNs from compact CPPN representations. The solution ANN with 40 hidden neurons and 356 connections is encoded by a much smaller CPPN with only 5 hidden neurons and 18 connections.

In contrast to direct encodings like NEAT [14, 16], genotypic CPPN mutations can have a more global effect on the expressed ANN patterns. For example, changes in only four CPPN weights are responsible for the change in topology from the second to the third ANN milestone. Other solutions followed similar patterns but single neuron or connection additions to the substrate do also sometimes occur.

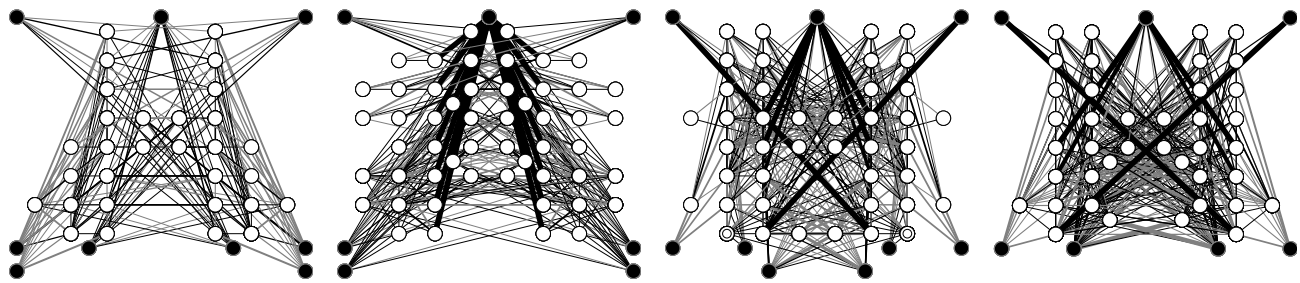
6. DISCUSSION

Previous work showed that ES-HyperNEAT and the original HyperNEAT exhibit similar performance in a simple navigation domain [10]. However, in the more complicated navigation domain presented here, the best fixed-substrate HyperNEAT method FS10x1 succeeds in only 45% of runs. How can this poor performance be explained?

Because of the increased complexity of the domain, it requires incrementally building on previously discovered stepping stones. While direct encodings like NEAT [14, 16] can complexify ANNs over generations by adding new nodes and connections through mutation, the indirect HyperNEAT encoding tends to start already with fully-connected ANNs [3], which take the entire set of ANN connection weights to represent a partial task solution. On the other hand, ES-HyperNEAT is able to elaborate on existing structure in the substrate during evolution (figure 8). This results is important because the more complicated the task, the more likely it will require a neuroevolution method that benefits from previously discovered stepping stones.

These results also explain why uniformly increasing the number of hidden nodes in the substrate does not necessarily increase HyperNEAT’s performance. In fact, FS8x8 performs significantly worse than FS5x5, which is likely due to the increased crosstalk that each neuron experiences.

The convention in HyperNEAT of the last several years was that the user would simply decide a priori where the hidden nodes belong. Yet, as the results presented here show, dictating that hidden nodes must exist at specific positions creates an unintentional constraint that any pattern of weights encoded by the CPPN must intersect those hidden node positions precisely with the correct weights. While



(a) Gen 24. ANN: 30 n, 184 c, (b) Gen 30 (ANN: 52 n, 280 c, (c) Gen 106 (ANN: 42 n, 310 c, CPPN: 3 n, 10 c, $f=0.85$) CPPN: 3 n, 10 c, $f=0.93$) (d) Gen 237 (ANN: 40 n, 356 c, CPPN: 5 n, 18 c, $f=10.00$)

Figure 8: ANN Milestones From a Single Maze Solution Lineage. Four ANN milestones are shown together with the number of hidden neurons n and connections c in the ANN and in the underlying CPPN. Fitness f is also shown. 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 gradual increase of connections indicates an increase of information in the hypercube, which in turn leads to an increase in performance.

ES-HyperNEAT can compensate for movement of information within the hypercube by expressing the hidden nodes at slightly different locations (e.g. figure 8c,d), representing the correct pattern for the original HyperNEAT is more difficult, resulting in more complex CPPNs. The significantly reduced performance of the vertical node arrangement FSLx10 in the maze navigation domain (figure 7a) confirms this hypothesis and shows that the more complex the domain the more restrictive it is to have nodes at fixed locations.

Finally, while on simple tasks it may sometimes incorporate more structure than necessary, such a capability may prove essential in more complex domains.

7. CONCLUSIONS

This paper presented a revision to ES-HyperNEAT that iteratively discovers the placement and density of the hidden nodes starting from the inputs and simultaneously from the outputs of the ANN based on implicit information in an infinite-resolution pattern of weights. The new approach significantly outperforms the original HyperNEAT with increased domain complexity because it can evolve ANNs with limited connectivity, elaborate on existing ANN structure, and compensate for movement of information within the hypercube. The main conclusion is that domains that require traversing many stepping stones that were too complex for regular HyperNEAT may now come within reach.

Acknowledgments

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

References

- [1] J. C. Bongard. Evolving modular genetic regulatory networks. In *Proceedings of the 2002 Congress on Evolutionary Computation*, 2002.
- [2] 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*, NJ, USA, 2009. IEEE Press.
- [3] J. Clune, B. E. Beckmann, P. McKinley, and C. Ofria. Investigating whether hyperneat produces modular neural networks. In *GECCO '10: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 635–642, New York, NY, USA, 2010. ACM.
- [4] R. Finkel and J. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.
- [5] D. Floreano, P. Dürri, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.
- [6] 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.
- [7] J. Gauci and K. O. Stanley. Autonomous evolution of topographic regularities in artificial neural networks. *Neural Comput.*, 22:1860–1898, 2010.
- [8] G. S. Hornby and J. B. Pollack. Creating high-level components with a generative representation for body-brain evolution. *Artificial Life*, 8(3), 2002.
- [9] J. Lehman and K. O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. In S. Bullock, J. Noble, R. Watson, and M. Bedau, editors, *Proceedings of the Eleventh International Conference on Artificial Life (Alife XI)*, Cambridge, MA, 2008. MIT Press.
- [10] S. Risi, J. Lehman, and K. O. Stanley. Evolving the placement and density of neurons in the hyperneat substrate. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)*, pages 563–570, 2010.
- [11] A. Rosenfeld. Quadrees and pyramids for pattern recognition and image processing. In *Proc. of the 5th Int. Conf. on Pattern Recognition*, pages 802–809. IEEE Press, 1980.
- [12] 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, NY, USA, 2008. ACM.
- [13] 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.
- [14] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.
- [15] K. O. Stanley and R. Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130, 2003.
- [16] K. O. Stanley and R. Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100, 2004.
- [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] P. Strobach. Quadtree-structured recursive plane decomposition coding of images. *IEEE Transactions on Signal Processing*, 39:1380–1397, 1991.
- [19] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9), September 1999.