Critical Factors in the Performance of HyperNEAT

Thomas van den Berg
Informatics Institute
University of Amsterdam
thomas.vandenberg@student.uva.nl

Shimon Whiteson
Informatics Institute
University of Amsterdam
s.a.whiteson@uva.nl

ABSTRACT

HyperNEAT is a popular indirect encoding method for evolutionary computation that has performed well on a number of benchmark tasks. This paper presents a series of experiments designed to examine the critical factors for its success. First, we determine the fewest hidden nodes a genotypic network needs to solve several of these tasks. Our results show that these tasks are easy: they can be solved with at most one hidden node and require generating only trivial regular patterns. Then, we examine how HyperNEAT performs when the tasks are made harder. Our results show that HyperNEAT's performance decays quickly: it fails to solve all variants of these tasks that require more complex solutions. Next, we examine the hypothesis that *fracture* in the problem space, known to be challenging for regular NEAT, can be even more so for HyperNEAT. Our results suggest that quite complex networks might be needed to cope with fracture and HyperNEAT can have difficulty discovering them. Finally, we connect these results to previous experiments showing that HyperNEAT's performance decreases on *irregular* tasks. Our results suggest irregularity is an extreme form of fracture and that HyperNEAT's limitations might be more severe than those experiments suggested.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Connectionism and neural nets*

General Terms

Algorithms, Experimentation, Performance

Keywords

HyperNEAT, Evolutionary Computation, Indirect Encoding

1. INTRODUCTION

Unlike traditional direct encodings in evolutionary computation, in which the genotype is identical or isomorphic to the phenotype, *indirect encodings* [1, 14, 19, 24] employ a more complex process to *develop* the phenotype from the genotype. Indirect encodings hold enormous promise for evolutionary computation because they enable the reuse of genes, i.e., each gene can be expressed in multiple places throughout the phenotype. As a result, not only do they encourage phenotypes with the regular structure often needed for complex tasks, they also have the potential for greater scalability, since the dimensionality of the phenotype can be far larger than that of the genotype [24].

However, getting indirect encodings to work in practice has proven challenging, and the best way to represent the mapping from genotype to phenotype has long been unclear. Fortunately, substantial progress has been made in the last few years. In particular, *HyperNEAT* [22] has recently become a popular indirect encoding method for neural networks. Unlike previous indirect encoding methods, which build the phenotype temporally, in step-by-step fashion [1, 14, 19], HyperNEAT "captures the essential properties of natural developmental encoding without implementing a process of development" [21]. In particular, it uses a genotypic *compositional pattern producing network* (CPPN) to draw the connectivity of the phenotypic *substrate network* on the inside of a hypercube. By exploiting a geometric representation of the space in which the substrate resides, HyperNEAT "can produce spatial patterns with important geometric motifs that are expected from generative and developmental encodings and seen in nature." [22].

HyperNEAT has often empirically outperformed direct encodings. It performed translation-invariant detection of objects [22] and discovered a controller for food-gathering [22] and line-following [12] robots. HyperNEAT also evolved the walking behavior of a quadruped robot [4], controllers for robot keepaway [26], produced an evaluation function for checkers [13], and simultaneously evolved controllers for multiple agents using a single genotype [9].

However, few studies have specifically examined *why* HyperNEAT succeeds in these tasks. Do these results in fact provide evidence that "symmetry, imperfect symmetry, repetition, and repetition with variation...are compactly represented and therefore easily discovered" [22]? This paper aims to help answer this question by examining the critical factors in the empirical performance of HyperNEAT on some of these tasks. Our primary goal is to ascertain to what degree the reasons for this success correspond with the motivating intuition behind HyperNEAT.

To this end, we first examine the complexity of several tasks in which HyperNEAT has succeeded. For each task, we establish an upper bound on complexity by determining the smallest number of hidden nodes the CPPN must have

GECCO'13, July 6–10, 2013, Amsterdam, The Netherlands.

to solve it. Surprisingly, our results indicate that all of these tasks are extremely easy once transferred to the connectivity hypercube encoding, as they can be solved by CPPNs with at most one hidden node. Consequently, they require generating only the most trivial regular patterns.

Next, we examine how HyperNEAT performs when the complexity of these tasks is increased. Unfortunately, these results indicate that HyperNEAT's performance decays quickly. In fact, it fails to solve all variants of these tasks in which a more complex—but still regular—solution is required.

In addition, we dig deeper into the cause of HyperNEAT's poor performance on more complex regular tasks by examining the hypothesis that *fracture* in the problem space, which is known to be challenging for the original NEAT method [16], can be even more problematic for HyperNEAT. Our results suggest that quite complex CPPNs are needed to cope with even modest fracture, and HyperNEAT can have difficulty discovering such CPPNs.

Finally, we connect these results to previous experiments showing that HyperNEAT's performance decreases as *irregularity* in the required solution increases [7]. Our analysis suggests that irregularity is actually an extreme form of fracture and that HyperNEAT's limitations are thus more severe than was suggested by those previous experiments.

2. BACKGROUND

NeuroEvolution of Augmenting Topologies (NEAT) is a method for using evolutionary computation to optimize neural networks [23]. It evolves not only the weights but also the topology of the network, by way of mutation operators that add new nodes and new connections. NEAT starts with a population of minimal topologies and adds complexity gradually through mutations, thereby keeping the search space small in early generations. Furthermore, NEAT adds *historical markings* to keep track of genealogies: during selection and reproduction, individuals with similar historical markers are put in the same *species*, wherein they share fitness. This speciation helps maintain population diversity.

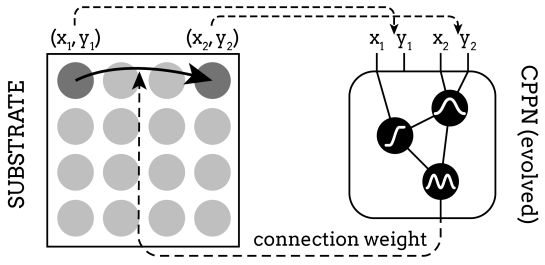

Figure 1: HyperNEAT evolves CPPNs that generate connection weights for the substrate networks applied to the task. The geometric locations of each pair of substrate nodes are fed to the CPPN to yield the weight of the connection between those nodes.

HyperNEAT is the indirect encoding extension of NEAT [22]. Instead of applying the evolved network directly to the task, HyperNEAT uses it as a *compositional pattern producing network* (CPPN), i.e., a genotype that defines the connections for a—possibly much larger—*substrate* network, i.e., the phenotype that is actually applied to the task. In particular, HyperNEAT expresses the connectivity of the substrate network in terms of the geometric location of its nodes, i.e., the substrate is represented by its *connectivity hypercube*. A CPPN is a way to represent this relation between node locations and connection weights. The CPPN is evolved by NEAT but the mutation operators are extended to allow different *node types*. Regular NEAT generates networks containing only sigmoid nodes, but HyperNEAT uses an expanded set: a sine, a bounded linear function, a Gaussian, a sigmoid, and an absolute value. A substrate network is constructed by querying the CPPN for the connection weight between each pair of nodes in the substrate, with those nodes' Cartesian coordinates as input. Thus, the CPPN is a compound function of pairs of node coordinates: $\text{CPPN}(x_1, y_1, x_2, y_2) = \text{weight}(n_1, n_2)$ (see Figure 1).

In this way, HyperNEAT tries to exploit the fact that, in many tasks, a human with some domain knowledge can often place substrate nodes in a space such that there is *regularity* in the desired connection weights. For example, it is natural to place the nodes representing a checkers board in a rectangular grid, so that HyperNEAT can exploit the fact that the board has two sides belonging to opposite players (symmetry), and that pieces can move to adjacent spaces throughout the board (repetition) [13].

3. RELATED WORK

Some previous research has also analyzed different factors in HyperNEAT's performance. In particular, [3] shows that HyperNEAT does not always generate modular solutions to problems that would benefit from modularity, and [27] shows that HyperNEAT can be modified to encourage modularity. These results are consistent with those we present in Sections 4 and 5, which suggest that HyperNEAT may be biased towards globally uniform connection patterns.

Other research showed that decreasing the *regularity* of the required solutions increases the difficulty for HyperNEAT, such that a direct encoding eventually performs better [5, 7]. Similarly, [6] examines the sensitivity of HyperNEAT to different arrangements of substrate nodes, showing that random configurations, by introducing irregularity, negatively impact performance, even though HyperNEAT still outperforms a direct encoding in some cases. In Section 6, we relate these effects to the concept of problem space *fracture*, which has been shown to be a critical factor in the performance of regular NEAT [16]. Our experiments regarding fracture confirm those of previous experiments showing that irregularity decreases performance, but also show that HyperNEAT fails even at moderate levels of fracture, which correspond to very low levels of irregularity. [2] replaces the NEAT part of HyperNEAT with genetic programming, which improved performance on the line following task; this approach is similar to our wavelet baseline (see Section 6) but was not tested on more difficult problems.

Finally, [28] demonstrates that the CPPNs generated under a "loose" fitness function (humans selecting which images they prefer) could not be reproduced by NEAT under a "tight" fitness function (minimizing distance to one specific image generated under the loose fitness function). On the one hand, these results are consistent with ours, as we also show (in Section 6) that HyperNEAT repeatedly fails when a specific nontrivial regularity is required. Our results show that this phenomenon also holds for much easier tasks of this type. On the other hand, we also show (in Section 5) that tasks with looser fitness functions, in the sense that they do not demand a specific target phenotype, can also be problematic for HyperNEAT, even when the tasks are only slightly harder than those considered in previous work.

4. COMPLEXITY

In this section, we examine the complexity of several tasks in which HyperNEAT has previously succeeded, where complexity is defined as the minimum number of CPPN hidden nodes required to solve the task. To do so, we apply a variant of HyperNEAT that limits the number of hidden nodes a CPPN can have: beyond that limit, node-insertion mutations are no longer executed. To establish an upper bound on a task's complexity, we apply this variant with the limit set to different levels. Unless stated otherwise, all results use the same parameter settings as the original published results and are averaged over 30 runs. The bars in each figure show the standard error for selected intervals. We used the *peas* implementation (`https://github.com/noio/peas/tags`) of HyperNEAT for the visual discrimination and line following tasks. For the walking gait task we used the original implementation (`http://jeffclune.com/research.html`).

4.1 Visual Discrimination

The visual discrimination task, one of the first to show HyperNEAT's effectiveness [22], mimics how the same pattern can be recognized equivalently at different places throughout the visual field. This is a kind of regularity: there should be a similar response at each location where the pattern appears and therefore similar substrate weights at each of these locations. The input to the task is an 11×11 image, containing a 3×3 *target square* and 1×1 *distractor square*. The goal is to "point out" the target square by giving the highest activation to the node at its center. Fitness is inversely proportional to the distance between the node with the highest activation and the target square's center. The substrate is a *sandwich network*: an input layer connects directly to an output layer, both matching the image size.

We use the *deltas* setting of the original task, so the CPPN receives an additional input of $x_1 - x_2$ and $y_1 - y_2$. In addition, in our experiment, the distractor object is placed randomly, in contrast to the original experiment, which placed it at a fixed location relative to the target square. Random placement ensures each target square location is paired with *different* distractor square locations during fitness evaluation, making the task more difficult.

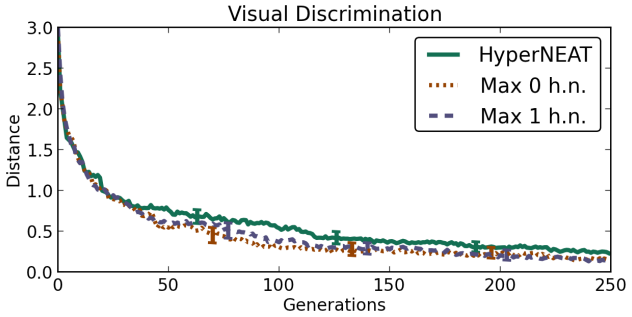

Figure 2: Average distance to target location of generation champions on visual discrimination task.

Figure 2 displays our results, which show that performance does not degrade when limiting HyperNEAT to CPPNs *without* hidden nodes. This is surprising, since such a CPPN can compute only single functions of a linear combination of its inputs. Examining these solutions reveals that they mostly rely on the fact that the target square has a larger area and thus a higher imprint on the substrate activation.

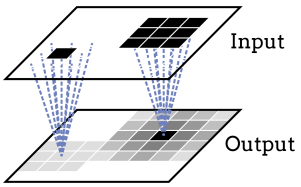

Figure 3: Solution to the visual discrimination task, showing incoming connections for two sample nodes. The target and distractor squares are activated and darker values indicate higher activation.

To accumulate this higher activation in the target location, each input node need only be connected to adjacent nodes in the output layer, as shown in Figure 3. Thus, as mentioned in [22], the task can be solved by any CPPN that gives high values to connections if $x_1 - x_2 \approx y_1 - y_2 \approx 0$. A function such as $\text{gauss}(w_1 x_1 + w_2 x_2 + w_3 y_1 + w_4 y_2)$ meets this requirement when $w_1 \approx -w_2$ and $w_3 \approx -w_4$, and this is exactly the function that nearly all the champions compute in runs where no hidden nodes are allowed. Hence, solving this problem requires optimizing only four weights. This kind of regularity, where connections are strong between adjacent nodes, is one of the most trivial, with only degenerate functions, e.g., outputting an equal weight for *every* connection, being simpler. By contrast, final champions of regular HyperNEAT runs used on average 11 hidden nodes to compute a similar function, suggesting substantial superfluous complexity.

4.2 Line Following

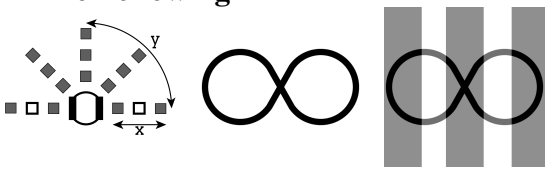

Figure 4: Left: sensor layout for line following task; middle: road that the robot must stay on; right: observation in the task variation from Section 5.2.

In the line following task, a simulated wheeled robot must drive on a flat terrain with zones of different friction (e.g., roads and grass) [2, 12]. The goal is to maximize average speed by staying on the roads, where the friction is lowest. The robot has 5×3 sensors mounted on antennae that can detect the type of terrain under them (see Figure 4). Two independently controlled wheels enable driving and steering. The task has two forms of regularity. First, the robot is symmetrical as the antennae are laid out radially and the wheels are opposite each other. Second, the five antennae are identical in length and response function.

The substrate consists of two layers: a bottom layer containing all the sensor inputs and a top layer containing hidden nodes and the output nodes that control the wheels. The CPPN is supplied with the angle and radius of each sensor location, scaled to a value between -1 and 1. The top layer has only 3×3 nodes; the output nodes are on the mid-left and mid-right, controlling the left and right wheel respectively. The other nodes in the top layer serve as hidden nodes. The CPPN has three output nodes, which specify connections from the bottom to the top layer, internal connections in the top layer, and the top layer bias, respectively.

Figure 5 shows our results. Though convergence is somewhat slower, final performance is just as good when Hyper-

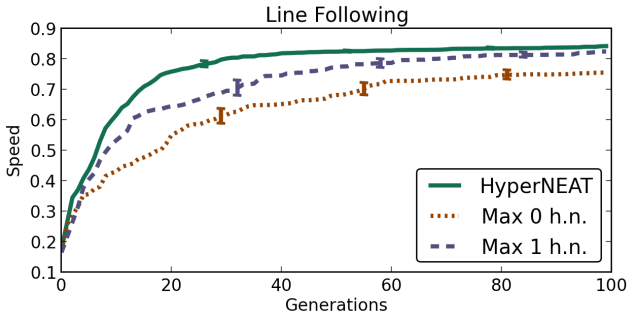

Figure 5: Average speed of generation champions on the line following task.

NEAT is restricted to one hidden node. Final performance is slightly lower without hidden nodes. By contrast, the champions at the end of the regular HyperNEAT runs had an average of 13 hidden nodes. Analyzing the evolved solutions reveals why the task is so easy: a high friction area observed on either side causes a high activation to input nodes on that side; by sending that high output to the wheel on that side, the robot steers in the *other* direction, away from it. Thus, the apparent symmetry need not be exploited explicitly. As with the visual discrimination task, solving it requires only that nodes be connected to others nearby, which is trivial in a connectivity hypercube.

4.3 Walking Gait

In the walking gait task, a four-legged table-shaped robot simulated in a physics engine has to walk as far as possible [4]. Since finding stable and energy efficient gaits for walking robots is known to be challenging, HyperNEAT's success on this task is considered a significant result. Each leg has three joints: two in the hip that allow anteroposterior and lateral motion of the upper leg and one in the knee for flexing of the leg. Values on the substrate's output layer determine target angles for each of the 3×4 joints. The input layer specifies the current angle of each joint, the angle of the body (roll, pitch, and yaw), whether each leg is touching the ground, and a time-based sine signal. The substrate has a single hidden layer and no recurrent connections. The task is regular because the legs have identical structure and likely need to behave similarly. While the walking behavior also displays *temporal* regularity due to the repeated motion, this periodicity is provided externally by the sine input.

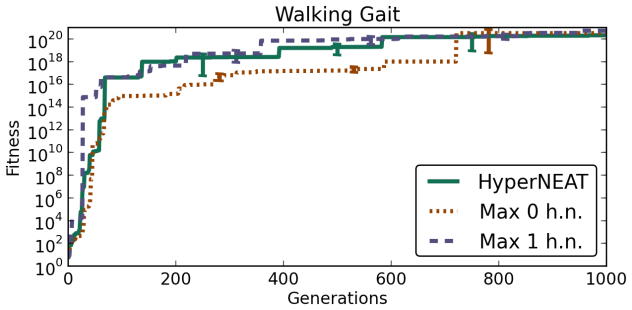

Figure 6: Average fitness of generation champions on the walking gait task.

Figure 6 displays our results, which show that this task can be solved by a CPPN without hidden nodes. Such a CPPN computes only a sigmoid function of a linear combination of its inputs. By contrast, the champions at the end of the regular HyperNEAT runs had an average of nearly 12 hidden nodes. Analyzing the solutions reveals that for most solutions, all joints move in synchrony with the supplied sine pulse, yielding a rigid hopping behavior. This explains why such a simple CPPN can solve the task: only the signal from the sine input needs to be sent to all the motor actuators. Generating the substrate connections using a connectivity hypercube leaves very little for NEAT to optimize.

5. SCALABILITY

The results presented above show that the considered tasks can be solved with trivial CPPNs. On the one hand, this is a success for HyperNEAT since the point of indirect encodings is to enable large phenotypes to be evolved using simple genotypes. HyperNEAT's encoding does so efficiently, since it dramatically reduces the number of parameters needed to solve these problems. On the other hand, it seems likely that, even after maximal compression, many realistic tasks require nontrivial genotypes. These results show that previously published HyperNEAT successes on the considered tasks do not confirm its ability to do so. In this section, we present results of experiments designed to fill this gap.

For each task, we apply HyperNEAT to the simplest extension that we could devise, in order to increment task difficulty only slightly. In addition, we compare the performance of HyperNEAT on these variations to that of a simple baseline algorithm that we devised. Note that the goal of this paper is not to contribute or advocate for any new method. On the contrary, this baseline is used only to establish that the task variants are not too difficult for *any* method: if HyperNEAT fails at the task variant but the baseline algorithm succeeds, then we know the results indicate a limitation of HyperNEAT rather than an excessively difficult task.

This baseline algorithm utilizes the same connectivity hypercube encoding, using the node coordinates to generate the connection weights. This is the part of HyperNEAT that is likely to be responsible for the good performance on the easy tasks, and so it is a good starting point for a comparison. However, it does not employ a CPPN to generate the connectivity but instead computes connection weights using a sum over a number of *wavelet* basis functions. A wavelet can be seen as a brief oscillation: it is periodic, but the amplitude starts out at zero, increases to a maximum, and then decreases. *Gabor wavelets*, often used in computer vision for image analysis, are among the simplest, as they are the product of a sine and a Gaussian [11, 17]. In the wavelet method, the connectivity C of nodes located at (x_1, y_1) and (x_2, y_2), i.e., the weight of their connection, is:

$$C((x_1, y_1, x_2, y_2) = \sum_{i=0}^{N} W_i \left(\begin{bmatrix} x_1 & y_1 & x_2 & y_2 & 1 \end{bmatrix} \right),$$

where W_i, the i-th wavelet, is:

$$W_i(\mathbf{x}) = \alpha_i \cdot G(\mathbf{x} \cdot \mathbf{a_i}, \mathbf{x} \cdot \mathbf{b_i}, \sigma_i).$$

Here, G is a single Gabor wavelet (see Figure 7):

$$G(x, y, \sigma) = \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \cos\left(2\pi x + \frac{\pi}{2}\right),$$

and the input coordinates are weighted by vectors $\mathbf{a_i}$ and $\mathbf{b_i}$.

Thus, each wavelet has the following parameters that must be optimized: $\alpha, \mathbf{a}_0, \ldots, \mathbf{a}_n, \mathbf{b}_0, \ldots \mathbf{b}_n, \sigma$. This is done with

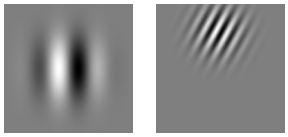

Figure 7: Left: a Gabor wavelet $G(x, y, \sigma)$ for $x, y \in [-2, 2]$ and $\sigma = 0.5$. Right $G(x', y', \sigma)$ with x' and y' obtained from an arbitrary linear transformation and $\sigma = 1.5$. Lighter colors indicate higher output values.

a simple evolutionary method employing two kinds of mutations. Either a new wavelet is added with probability 0.1, or, for each wavelet, the parameters are perturbed with probability 0.3. When a new wavelet is added, it is initialized with a value drawn from a normal distribution $\mathcal{N}(0, 0.1)$ for all parameters. When a wavelet is perturbed, $\mathcal{N}(0, 0.1)$ is added to each parameter. Tournament selection ($k = 3$) is employed and each generation champion is carried over unmodified to the next generation. The population size corresponds to the HyperNEAT settings for each experiment.

5.1 Visual Discrimination

The original visual discrimination task could be solved by spreading out activation across the substrate and relying on the larger area of the target square to increase activation of the correct output. To make this task more difficult, we turn it into a true *shape discrimination* task, similar to [25], where the target and distractor objects differ only in form, not size. Regularity is preserved because the same objects still need to be recognized across the entire visual field. In addition, the target and distractor objects are now mirror images of each other: they are both triangles created by diagonally slicing the original target square. The center location is still at the center of the bounding square, as indicated by the underlined entry below:

$$\text{target} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & \underline{1} & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{distractor} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

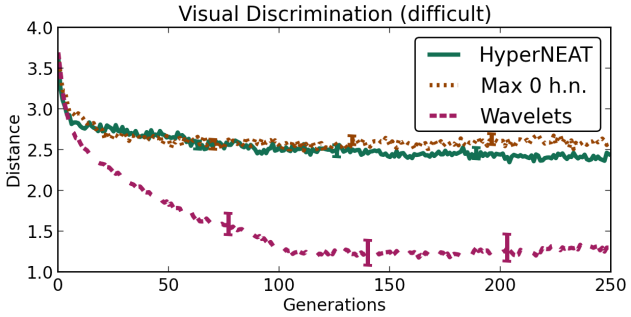

Figure 8: Average distance to target of generation champions on the difficult visual discrimination task.

Figure 8 shows our results. The average distance to the center node is 3.4; this is the score obtained when the network always activates the node at the center of the visual field, regardless of input. When the generated substrate does not discriminate between the shapes, and simply picks the center of mass of the objects combined, a strategy similar to that in the previous experiment, the average distance is about 2.5. HyperNEAT performs slightly better than that but again fails to substantially outperform the variant forbidden to use hidden nodes. The wavelet baseline method performs much better and the fact that the average distance is below 2.5 indicates that it is indeed discriminating between the two shapes and that the task is thus solvable.

5.2 Line Following

To make the line following task more difficult, we add a discrepancy between the observation and the actual friction value of the terrain. The road still looks darker than its surroundings, but the actual values are not the same everywhere, as shown in the right part of Figure 4. To solve this task, the substrate has to *compare* sensor values in order to find out whether they correspond to the road. In addition, the robot has to accumulate these values directly into the output nodes because recurrent connections are disabled, adding extra difficulty to the task. Regularity is still present, as the layout of the robot remains unchanged.

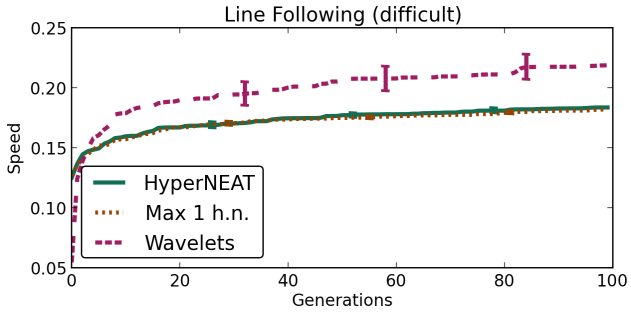

Figure 9: Average speed of champions on the difficult line following task.

Figure 9 shows our results. Both regular HyperNEAT and the variant restricted to one hidden node perform equally poorly, while the wavelet method attains a substantially higher speed. Examination of the resulting controllers shows that those generated by HyperNEAT frequently adjust their trajectory only when they are already off-road. The wavelet method's performance shows that avoiding this behavior and staying consistently on the road is feasible.

5.3 Walking Gait

In the original walking gait task, many runs evolve the simplest solution: a hopping behavior where all legs move together. This is efficient because each hop propels the robot a large distance. However, in real robots, actuators are typically much less powerful compared to the robot's weight. To create a more realistic simulation where hopping is *not* efficient, we increased the mass of the torso in the physics engine from 10 to 100 units. This decreases the relative force exerted by the joint motors, with two consequences. First, hopping is not as effective since the robot cannot hop very far. Second, it is less stable, as it cannot always absorb the full impact of landing on a single leg, making balance more important. The regularity of the task remains unaltered.

Figure 10 shows our results. As expected, the HyperNEAT robots cover substantially less distance than in the original task. Analyzing the resulting gaits reveals that HyperNEAT still generates hopping gaits with similar outputs for every joint angle controller. While this makes sense for the anteroposterior joints, the lateral joints *also* receive a target angle, moving them sideways during the hop. This leaves the robot unbalanced and, because the joint motors have limited power, it sometimes falls over.

Of course, since the heavier torso will affect even good

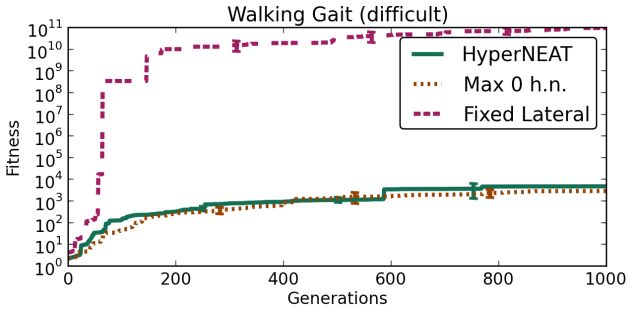

Figure 10: Average distance covered by champions on the difficult walking gait task.

walking gaits, it is important to determine whether better performance in this task variation is possible. Unfortunately, due to limitations in the software implementation created in [4], we were unable to implement the baseline wavelet method in this task. Instead, we constructed an alternative task-specific baseline as follows. HyperNEAT evolves the controller but, in order to generate more stable gaits, the lateral joints are fixed by setting the target angle to zero after the substrate has been queried for an output.

As shown in Figure 10, this baseline greatly outperforms regular HyperNEAT. In principle, regular HyperNEAT should be able to evolve CPPNs that generate *exactly* this behavior. In fact, doing so requires only removing the connections to all of the lateral weights, as shown in Figure 11. The failure of HyperNEAT to do so demonstrates that it does not account for this exception and that sideways motion of the lateral joints is an adverse effect of an overly simple solution.

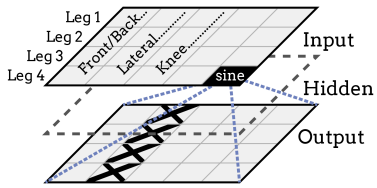

Figure 11: In the difficult walking gait problem, disconnecting the lateral joint nodes, indicated by crosses, improves performance. HyperNEAT could theoretically isolate these nodes but fails to do so.

6. FRACTURE

In this section, we examine the hypothesis that *fracture* in the problem space, which is known to be challenging for the original NEAT method [16], can be even more problematic for HyperNEAT. For regular NEAT, fractured problems are defined as those with "a highly discontinuous mapping between states and optimal actions" [16]. To quantify this notion, [16] expresses it in terms of *function variation* in the output of the network generated by NEAT, measured as the sum of the differences of adjacent values on a sampled version of the function. For example, for a one-dimensional function, if $X_{0,\ldots,n}$ is a set of n sample points on the domain of function f, the fracture is $\sum_{i=0}^{n-1} |f(X_i) - f(X_{i+1})|$.

However, this definition is not appropriate for a CPPN because even the simplest CPPNs can include a sine node which, at the right frequency, would cause maximal function variation. To more accurately capture the essence of fracture as the degree to which regions of the generated function have to be treated differently, we propose an alternative definition of fracture for indirect encodings. Because nodes in HyperNEAT can actually emit repeated patterns, we define fracture as a *discontinuous variation of patterns*. If we are free to construct the target function that the CPPN is to approximate, we can formalize this as the number of contiguous and convex *regions* in the target function with different patterns (see Figure 12). Because it is hard to distinguish between regions and patterns in an arbitrary problem, we use this only as a *generative* definition: delineating regions in problems for which we construct the target function.

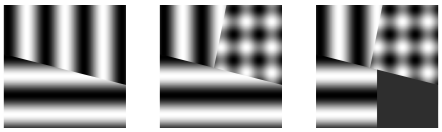

Figure 12: Increasingly fractured spaces.

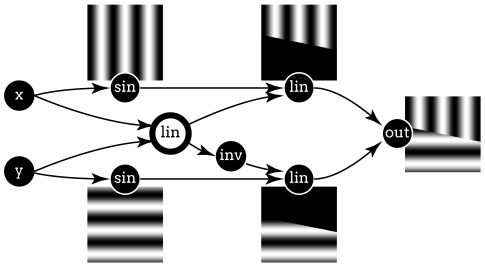

Figure 13: Combining two patterns into a fractured substrate: the outlined node forms a linear combination of the coordinate vector that emits a *mask* indicating whether (x, y) is in the target region; regions are zeroed out by subtracting this value from the pattern, since values below some threshold are truncated; finally, the different patterns are summed.

The reason we hypothesize that fracture can be difficult for HyperNEAT is that it seems that many nodes are needed to divide the substrate into such regions. Though the CPPN can evolve nodes for generating different patterns, some mechanism is needed to select which pattern applies in which region. An *indicator node* is needed that specifies whether the given (x, y) is inside or outside the region, which is complex since HyperNEAT needs multiple nodes to compute a nonlinear combination of the coordinates. The simplest solution would be to *multiply* the value of indicator node by that of the pattern, yielding the pattern inside the region, and zeroes outside of it. This is impossible, however, because the CPPN lacks a multiplication operator.

Workarounds are possible but they are complex. Figure 13 illustrates the simplest workaround we could devise for a substrate divided along an infinite line. Even more nodes are needed to isolate finite convex regions, and concave regions must be split up into multiple convex ones. Since CPPNs with only some of the nodes required for this workaround are likely to perform poorly, we hypothesize that it is difficult for HyperNEAT to evolve such workarounds incrementally.

6.1 Target Weights

To determine the effects of *fracture* on HyperNEAT's performance, we use the target weights task [7], whose goal is to generate a specific weight pattern in the substrate. It was originally used to show that performance of HyperNEAT degrades as *irregularity* increases. Irregularity was increased

by adding noise: an increasing percentage of randomly chosen weights were each assigned a different random value. This can be seen as an extreme form of fracture, since a unique value has to be assigned to every region consisting of a single connection weight, unless adjacent connections form contiguous (patterned) regions by chance. The substrate in the original experiment consists of 3^4 connections so, even at 10% noise, approximately eight new regions are introduced.

Therefore, if our hypothesis is correct, we should expect that HyperNEAT would perform poorly even at low noise levels. To assess the quality of HyperNEAT's solutions, we compare to a baseline consisting of a linear least-squares solution for each problem instance, the coefficients of which are the node coordinates that are also input to the CPPN. Note that this is a very weak baseline, as a linear solution cannot cope with *any* of the irregularity or fracture.

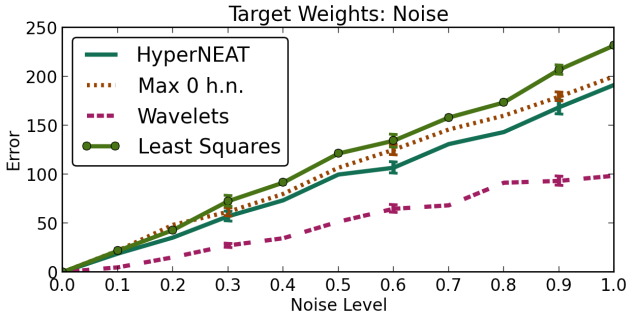

Figure 14: Average error of final generation champions in the target weights task with increasing noise, averaged over 10 runs, as in the original experiment.

Figure 14 shows our results, which reproduce the original experiment but add the HyperNEAT variant restricted to no hidden nodes, the wavelet baseline method, and the linear least-squares solution. To be consistent with the least-squares solution, we base the fitness function on the sum of squared errors between the generated substrate and the target weights. In addition, for brevity, we show only performance of the champion of the last generation of each run, grouped by noise level. These results show that HyperNEAT is never doing much better than a linear solution, even at 10% noise, and is thus uniformly unable to account for the aberrant regions. Thus, while the increasing magnitude of error implies a gradual decline, HyperNEAT actually fails to model the noise at *any* level, likely because it cannot cope with the fracture such noise introduces. The wavelet baseline method, by contrast, substantially outperforms the linear solution.

6.2 Target Weights with Bisection

To determine what level of fracture HyperNEAT can cope with, we devised a second target weights experiment. Instead of assigning a random value to single weights, we divide the substrate into regions along straight lines, each time bisecting previous regions, as illustrated in Figure 15. Each of these regions is then assigned a single random weight. We decrease the dimensionality of the target matrix from 3^4 to 8^2 and continue splitting until each of the 8×8 connections consists of its own region. The result is that fracture increases gradually, enabling us to test fracture levels lower than that of the lowest noise level in the original experiment.

Figure 16 shows the results, which demonstrate that, even at very low levels of fracture, HyperNEAT cannot substantially outperform a linear solution, though the wavelet baseline can. Together, these results confirm our hypothesis that fracture can be highly problematic for HyperNEAT.

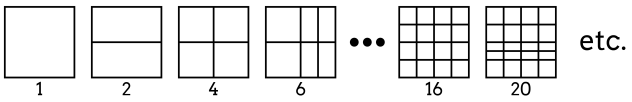

Figure 15: Iterative addition of fracture.

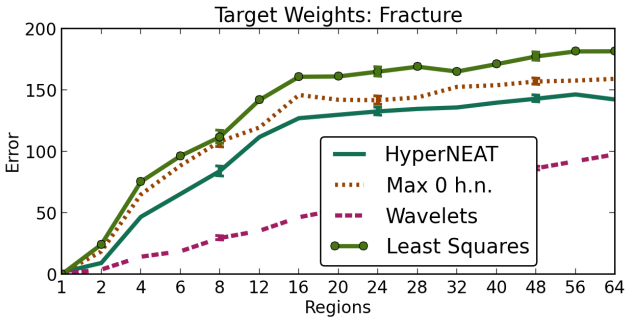

Figure 16: Average error of the final generation champions in the target weights task with increasing bisection, averaged over 100 runs.

7. DISCUSSION & FUTURE WORK

The results presented in this paper show that the success of HyperNEAT on the tasks we considered does not provide much support for the hypothesis that "symmetry, imperfect symmetry, repetition, and repetition with variation...are compactly represented and therefore easily discovered" [22]. On the one hand, on the simple versions of the considered tasks, HyperNEAT succeeds because the connectivity hypercube encoding represents simple regularity in such a way as to make the problem solvable by a trivial CPPN. On the other hand, on only slightly harder versions of these tasks, HyperNEAT does not enable the easy discovery of the more elaborate–but still regular–phenotypes these tasks require, as it is not able to evolve the corresponding genotypes.

This result is somewhat surprising, since incrementally evolving complex solutions is supposed to be NEAT's specialty. However, it seems that doing so in HyperNEAT is more difficult than in regular NEAT, though more research is needed to determine with certainty why that is.

There is some evidence that *novelty search* can improve the performance of NEAT [18] and HyperNEAT [20]. However, applying novelty search requires some domain knowledge to design an effective *behavior function* [15].

A caveat of our results in Section 6 is that, since each target weights task requires a specific phenotype to solve, they consider only tight fitness functions like those in [28]. This is a necessary limitation due to the generative nature of our definition of fracture. There is some evidence that HyperNEAT can cope with modest fracture (two regions) given looser fitness functions [8]. However, in our experiments, HyperNEAT could not solve the harder walking gait task, which also has a looser fitness function, even though it requires generating a pattern with only three regions (see Figure 11). More research is needed into the interplay between the tightness of the fitness function and HyperNEAT's ability to cope with fracture, though doing so is difficult since these parameters cannot be directly controlled.

Of course, a key limitation of all our results is that they consider only some of the tasks on which HyperNEAT has

succeeded. Thus, another important avenue for future work is to repeat this sort of analysis on these other tasks. While we can only speculate about HyperNEAT's behavior on such tasks, there are hints to suggest it is in at least some cases consistent with what we report here. For example, in checkers [13], HyperNEAT found a winning solution in only 8.2 generations on average, which may indicate that only a trivial CPPN is required. In keepaway [26], performance is competitive with other leading methods but the comparison is confounded by the use of a novel "bird's eye view" representation. It may be that this representation obviates the need to discover a nontrivial CPPN. Other HyperNEAT successes may be more substantial, e.g., it solved a harder visual discrimination task somewhat similar to ours in Section 5 [8], evolved elaborate behaviors for teams of robots [9, 10], and discovered more intricate walking gaits for real robots [29]. However, whether these tasks actually require complex solutions is not known.

To better cope with fracture, it might be beneficial to introduce nodes with a radial basis function, creating "RBF-HyperNEAT", corresponding to "RBF-NEAT" [16], which was shown to handle fracture better than regular NEAT. From there it is a small step to implement nodes emitting *wavelets*, which —given the results in this paper— could be an effective extension. Alternatively, the wavelet baseline method considered here itself showed promising initial performance and could be used as a starting point for a more robust indirect encoding method.

8. CONCLUSION

This paper studied the critical factors in the performance of HyperNEAT. We examined the difficulty of several tasks on which HyperNEAT has succeeded and found that all of these tasks are easy in the sense that they can be solved with at most one hidden node and require generating only trivial regular patterns. Then, we examined how HyperNEAT performs when the tasks are made harder and found that HyperNEAT's performance decays quickly: it fails to solve all variants of these tasks that require more complex —but still regular—solutions. Finally, we examined the role of problem space fracture and found that, in these experiments, HyperNEAT has trouble with even modest levels of fracture. Together, these results suggest that the capacity of HyperNEAT to capture complex regularities may be less than was previously supposed and hence new methods may be needed to finally fulfill the promise of indirect encodings.

9. REFERENCES

[1] J. Astor and C. Adami. A developmental model for the evolution of artificial neural networks. *Artificial Life*, 6(3):189–218, 2000.

[2] Z. Buk, J. Koutník, and M. Šnorek. NEAT in HyperNEAT substituted with genetic programming. *Adaptive and Natural Computing Algorithms*, pages 243–252, 2009.

[3] J. Clune, B. Beckmann, P. McKinley, and C. Ofria. Investigating whether HyperNEAT produces modular neural networks. In *GECCO*, pages 635–642, 2010.

[4] J. Clune, B. Beckmann, C. Ofria, and R. Pennock. Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In *Congress on Evolutionary Computation*, pages 2764–2771, 2009.

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

[6] J. Clune, C. Ofria, and R. Pennock. The sensitivity of HyperNEAT to different geometric representations of a problem. In *Conference on Genetic and Evolutionary Computation*, pages 675–682. ACM, 2009.

[7] J. Clune, K. Stanley, R. Pennock, and C. Ofria. On the performance of indirect encoding across the continuum of regularity. *Transactions on Evolutionary Computation*, 15(3):346–367, 2011.

[8] O. Coleman. Evolving neural networks for visual processing. *B.S. Thesis*, 2010.

[9] D. B. D'Ambrosio, J. Lehman, S. Risi, and K. Stanley. Evolving policy geometry for scalable multiagent learning. In *AAMAS*, volume 1, pages 731–738, 2010.

[10] D. B. D'Ambrosio, J. Lehman, S. Risi, and K. O. Stanley. Task switching in multirobot learning through indirect encoding. In *IROS*, pages 2802–2809, 2011.

[11] J. Daugman et al. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *Journal of the Optical Society of America A*, 2:1160–1169, 1985.

[12] J. Drchal, J. Koutník, and M. Snorek. HyperNEAT controlled robots learn how to drive on roads in simulated environment. In *CEC*, pages 1087–1092, 2009.

[13] J. Gauci and K. Stanley. A case study on the critical role of geometric regularity in machine learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, pages 628–633, 2008.

[14] G. Hornby, J. Pollack, et al. Body-brain co-evolution using L-systems as a generative encoding. In *GECCO*, pages 868–875, 2001.

[15] S. Kistemaker and S. Whiteson. Critical factors in the performance of novelty search. In *GECCO*, p.965–972, 2011.

[16] N. Kohl and R. Miikkulainen. Evolving neural networks for strategic decision-making problems. *Neural Networks*, 22:326–337, 2009.

[17] T. Lee. Image representation using 2D Gabor wavelets. *Transactions on Pattern Analysis and Machine Intelligence*, 18(10):959–971, 1996.

[18] J. Lehman and K. O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. *Artificial Life*, 11:329, 2008.

[19] A. Lindenmayer. Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. *Journal of Theoretical Biology*, 18(3):280–299, 1968.

[20] G. Morse, S. Risi, C. R. Snyder, and K. O. Stanley. Single-unit pattern generators for quadruped locomotion. In *GECCO*, 2013.

[21] K. Stanley. Exploiting regularity without development. In *Proceedings of the AAAI Fall Symposium on Developmental Systems*, 2006.

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

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

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

[25] M. Suchorzewski and J. Clune. A novel generative encoding for evolving modular, regular and scalable networks. In *GECCO*, pages 1523–1530, 2011.

[26] P. Verbancsics and K. Stanley. Evolving static representations for task transfer. *The Journal of Machine Learning Research*, 11:1737–1769, 2010.

[27] P. Verbancsics and K. Stanley. Constraining connectivity to encourage modularity in HyperNEAT. In *GECCO*, 2011.

[28] B. Woolley and K. Stanley. On the deleterious effects of a priori objectives on evolution and representation. In *GECCO*, pages 957–964, 2011.

[29] J. Yosinski, J. Clune, D. Hidalgo, S. Nguyen, J. Zagal, and H. Lipson. Evolving robot gaits in hardware: the HyperNEAT generative encoding vs. parameter optimization. In *ECAL*, volume 8, page 12, 2011.