# The Sensitivity of HyperNEAT to Different Geometric Representations of a Problem

Jeff Clune
Michigan State University
East Lansing, MI, USA
jclune@msu.edu

Charles Ofria
Michigan State University
East Lansing, MI, USA
ofria@msu.edu

Robert T. Pennock
Michigan State University
East Lansing, MI, USA
pennock5@msu.edu

## ABSTRACT
HyperNEAT, a generative encoding for evolving artificial neural networks (ANNs), has the unique and powerful ability to exploit the *geometry* of a problem (e.g., symmetries) by encoding ANNs as a function of a problem's geometry. This paper provides the first extensive analysis of the sensitivity of HyperNEAT to different geometric representations of a problem. Understanding how geometric representations affect the quality of evolved solutions should improve future designs of such representations. HyperNEAT has been shown to produce coordinated gaits for a simulated quadruped robot with a specific two-dimensional geometric representation. Here, the same problem domain is tested, but with different geometric representations of the problem. Overall, experiments show that the quality and kind of solutions produced by HyperNEAT can be substantially affected by the geometric representation. HyperNEAT outperforms a direct encoding control even with randomized geometric representations, but performs even better when a human engineer designs a representation that reflects the actual geometry of the robot. Unfortunately, even choices in geometric layout that seem to be inconsequential *a priori* can significantly affect fitness. Additionally, a geometric representation can bias the *type* of solutions generated (e.g., make left-right symmetry more common than front-back symmetry). The results suggest that HyperNEAT practitioners can obtain good results even if they do not know how to geometrically represent a problem, and that further improvements are possible with a well-chosen geometric representation.

## Categories and Subject Descriptors
I.2.6 [**Artificial Intelligence**]: Learning-Concept Learning, Connectionism and Neural Nets

## General Terms
Experimentation, Algorithms

## Keywords
HyperNEAT, NEAT, Neuroevolution, Representation, Geometry, Generative, Developmental, and Indirect Encodings, ANNs.


*GECCO'09*, July 8–12, 2009, Montréal, Québec, Canada.


## 1. INTRODUCTION AND BACKGROUND
Many problems tackled by the field of artificial intelligence have geometric regularities that are intuitively obvious to a human observer. For example, in the game of checkers the geometric concept of adjacency is important because pieces close together are likely to constrain each other. The edge squares on the top and bottom of the board are different because they can confer kingship, and all of the edge squares are important because a piece on them cannot be jumped. There are also symmetries to the game (e.g., left-right, top-bottom, rotational). Many other AI problems, from machine vision to robotic control, also contain a plethora of geometric information. Although geometric information could be helpful in solving these types of problems, most evolutionary algorithms do not make such geometric information available to be exploited [14]. It is common to provide sensory information, such as whether a piece is present in a square, without also specifying the associated geometric coordinates. Stripping out such information is akin to asking a human to learn to play checkers by cutting up the board into its constituent pieces and scattering them randomly on the floor [4].

A technique was recently introduced that provides geometric information to an evolutionary algorithm. It is a generative encoding called HyperNEAT that evolves ANNs [14]. HyperNEAT's ability to exploit the geometry of a problem has been repeatedly demonstrated [1-4, 14]. HyperNEAT's generative nature, wherein an element of a genome encodes for multiple elements in a phenotype, enables it to create phenotypic solutions that exhibit symmetries and repeated themes, with and without variation. These attributes are important when exploiting geometric information because geometric information is frequently symmetric and repetitive.

The ability to inject geometric information into an evolutionary algorithm necessitates that the experimenter choose how to represent that information. This requirement raises the question of how sensitive HyperNEAT is to different geometric representations of the same problem. There are aspects of some problems that have obvious geometrical representations (e.g., the Cartesian coordinate system of checkers), but other aspects of those problems may have no obvious geometric location. For example, where, geometrically, should the input for the current number of black pieces be placed? Other problems have many seemingly good geometric representations. For example, there are multiple ways to order the legs of a quadruped robot in two dimensions, and there are pluses and minuses for each alterative.

One possibility is that HyperNEAT is mostly immune to variation in the geometric representation. Such insensitivity would eliminate the need to spend time trying to select the most advantageous representation. It would also be surprising,

however, because preserving meaningful correlations in the problem's representation should aid HyperNEAT's performance. At the other extreme, it could be the case that HyperNEAT exhibits vastly different performance levels in response to even small changes in the geometric representation of aspects of the problem that seem to have no obvious geometric location in the representation (e.g., a piece counter). If the location of such information does matter, then HyperNEAT users should be aware of that fact so they know to try a variety of configurations.

This paper will investigate HyperNEAT's sensitivity to geometric representations by performing repeated tests of the same problem (with the same inputs and outputs), but with different representations of the geometric information. The problem domain will be that of evolving a controller that produces a walking gait for a simulated quadruped robot [1]. This domain is appropriate for this study for two reasons. Initially, some aspects of the problem, such as the ordering of joints in each leg, have clear geometric relationships, yet other aspects of the problem do not. For example, each of the knee joints should be geometrically represented as further from the torso than the hip joints, but other inputs (described below) have no obvious geometric location. Secondly, it has been previously shown that HyperNEAT is successful on this task with one specific geometric representation [1], providing a baseline performance that can be compared to the performance of other geometric representations.

## 2. PREVIOUS WORK

The authors are aware of only one algorithm besides HyperNEAT that injects geometric information into an evolutionary algorithm. The *simple geometry-oriented cellular encoding* (SGOCE) is a generative encoding that evolves ANNs that control legged robots [8]. Controllers were successfully evolved with SGOCE that could walk, follow gradients, and avoid obstacles. Unfortunately, the benefit of the geometric information in the SGOCE system was not isolated and investigated. While the system as a whole performed well on the legged-robot problem, alternate geometric configurations were not tested. Additionally, the cellular-encoding method that SGOCE was based on had been previously shown to perform well on the problem of evolving controllers for legged robots without the addition of geometry [5].

HyperNEAT has been demonstrated to exploit geometric information on a variety of tasks. It was shown to discover numerous different geometric motifs that it repeated to solve a visual discrimination task [14]. Its ability to exploit geometry was also verified in the checkers domain, where it evaluated board configurations [4]. HyperNEAT also exploited the geometric representation of a team of agents to produce herding strategies [3]. For example, left-right symmetries were discovered wherein the left and right groups of agents would perform strategies that were mirror images of each other. HyperNEAT has also been demonstrated to exploit geometric information to produce symmetries on the problem domain of this paper, that of evolving gaits for simulated quadruped robots [1]. For example, HyperNEAT frequently produced gaits that were symmetric with respect to each leg, such that all four legs moved in synchrony (four-way symmetry).

Only one published experiment tested HyperNEAT's sensitivity to alternate geometric representations of the same problem [14]. A simulated circular robot was rewarded for finding food. Eight food sensors and eight motor outputs were evenly distributed around the robot's center. The robot could move in the direction of any of those food sensors by activating the corresponding motor output. Two geometric configurations were tested: a *parallel* Cartesian coordinate configuration, where corresponding food sensors and motor outputs were labeled with the same Y coordinates but different X coordinates, and a *concentric* configuration, with a polar coordinate system, where corresponding food and motor nodes shared the same angle from 0°. Unless extra information about the distance between nodes was provided, the parallel configuration evolved strategies that collected food faster than the concentric configuration. Even though only one experiment was conducted, and it compared only two alternate geometric representations, the fact that the geometric representation made a difference is interesting and invites a more rigorous study of HyperNEAT's sensitivity to geometric representations. Such a study is conducted in this paper.

With respect to the problem domain of this paper, researchers have previously evolved controllers for legged robots with generative encodings [5, 6, 10, 11, 15], including HyperNEAT [1]. The intent of this paper is not to produce better solutions to this problem, but instead to utilize it as a means of testing the sensitivity of HyperNEAT to geometric representations. That said, HyperNEAT did produce impressive robotic gaits that were fast, natural, and graceful [1]. While the field lacks the benchmarks necessary to compare gaits, HyperNEAT can be considered a cutting edge algorithm for evolving gaits for legged robots. Learning more about how HyperNEAT is helped or hurt by varying its geometric representation should assist those wishing to capitalize on HyperNEAT's substantial potential in this domain.

## 3. THE EXPERIMENTAL SYSTEM

### 3.1 HyperNEAT

HyperNEAT [14] is a generative encoding that evolves ANNs with the principles of the widely used NeuroEvolution of Augmenting Topologies (NEAT) algorithm [12][1]. HyperNEAT evolves Compositional Pattern Producing Networks (CPPNs) [13], each of which is a function that takes inputs and produces outputs. If the goal is to evolve two-dimensional pictures, then the inputs to the CPPN function are the Cartesian coordinates of each of the pixels on the canvas. The output of the function determines the color of the pixel (Fig. 1).

Evolution modifies the population of CPPN functions. Each CPPN is itself a directed graph network, where each node is a math function such as sine or Gaussian. The nature of the functions included can create a variety of desirable properties, such as symmetry (e.g., an absolute value or Gaussian function) and repetition (e.g., a sine or cosine function) that evolution can take advantage of. Nested coordinate frames can develop in the directed network. For instance, a sine function early in the network can create a repeating theme that, when passed into the symmetric Gaussian function, creates a repeating series of symmetric motifs (Fig. 1). This process is similar to how natural organisms develop. For example, many organisms set up a repeating coordinate frame (e.g., body segments) within which are symmetric coordinate frames (e.g., left-right body symmetry). Asymmetries can be generated by sourcing global coordinate

[1] HyperNEAT is freely available (http://eplex.cs.ucf.edu). This description of HyperNEAT is adapted from [2].

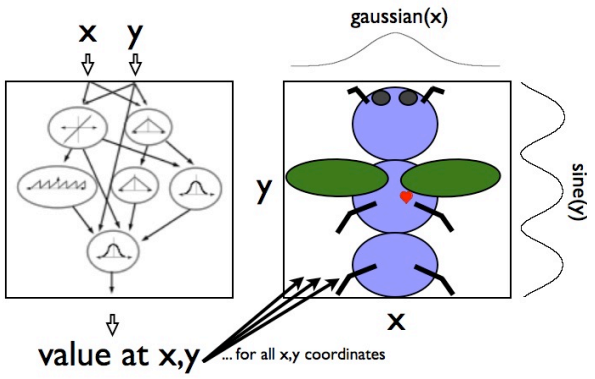


Figure 1. CPPNs can compose math functions to generate the properties of symmetry and modular repetition, with and without variation. This figure is adapted from [13].

frames such as f(x). The links between nodes in a CPPN have a weight value that can magnify or diminish the values that pass along them. Mutations that change these weights may, for example, give strong influence to a symmetry-generating part of a network while making the contribution from another part of the network more subtle. When CPPNs are evolved artificially with humans performing the selection, the evolved shapes look surprisingly beautiful, complex and natural [13, 18]. More importantly, the evolved shapes exhibit the desirable features of generative encodings, namely, the repetition of themes, symmetries, and hierarchies, with and without variation.

In addition to images, CPPNs can generate ANNs [14]. In this case, the inputs are a constant bias value and the locations on a Cartesian grid of both the source (e.g., $<x_1=4, y_1=4>$) and target (e.g., $<x_2=5, y_2=5>$) nodes. The function takes these five values (bias, $x_1$, $y_1$, $x_2$, $y_2$) as inputs and produces two output values [4]. The first value determines the weight of the link between the associated input (source) and hidden layer (target) nodes. The second value determines the weight of the link between the associated hidden (source) and output (target) layer nodes. All pairwise combinations of source and target nodes are iteratively passed as inputs to a CPPN to determine what the weight value is for each possible link. Thus, the CPPN function is a genome that encodes for an ANN phenotype (also called a *substrate*) [14].

A benefit of HyperNEAT is that it is capable of exploiting the geometry of a problem [14]. Because the link values between nodes in the final ANN substrate are a function of the geometric positions of those nodes, if those geometric positions represent aspects of the problem that are relevant to its solution, then HyperNEAT can exploit such information. For example, when playing checkers, the concept of adjacency (on the diagonals) is important. Link values between adjacent squares may need to be very different than link values between distant squares. HyperNEAT can exploit adjacency to create a connectivity motif and repeat it across the board [4, 14]. In the case of quadruped locomotion, HyperNEAT could, for example, implement front-back, left-right, or diagonal symmetries to produce common gaits.

Variation in HyperNEAT occurs when mutations change the CPPN function networks. Mutations can add a node to the graph, which results in the addition of a function to the CPPN network, or change its link weights. The functions in CPPNs in this paper are the standard set [14]: sine, sigmoid, linear, and Gaussian. The evolution of the population of CPPN networks occurs according to the principles of NEAT, which was originally designed to evolve ANNs. NEAT can be fruitfully applied to CPPNs because the population of CPPN networks is similar in structure to a population of ANNs.

The NEAT algorithm is unique in three main ways [12]. Initially, it starts with small genomes that encode simple networks and slowly *complexifies* them via mutations that add nodes and links to the network. This complexification enables the algorithm to evolve the network topology in addition to its weights. Secondly, NEAT has a fitness sharing mechanism that preserves diversity in the system and gives time for new innovations to be tuned by evolution before competing them against rivals that have had more time to mature. Finally, NEAT tracks historical information to perform crossover in a way that is effective, yet avoids the need for expensive topological analysis. A full explanation of NEAT can be found in [12].

## 3.2 FT-NEAT, a direct encoding control

An appropriate control for HyperNEAT is to evolve the weights for the same ANN substrate topology that HyperNEAT produces with a direct encoding that cannot exploit geometry. *Fixed Topology NEAT* (FT-NEAT) [1] is the same as HyperNEAT in all ways, except for the generative CPPNs. An instance of FT-NEAT that did not have hidden nodes, and was thus called *Perceptron NEAT* (P-NEAT), has previously served as a direct encoding control for HyperNEAT [2, 4, 14]. FT-NEAT is the same as NEAT without the complexification. The rest of the elements from NEAT (e.g., its crossover and diversity preservation mechanisms) remain the same between HyperNEAT and FT-NEAT, making FT-NEAT a good control.

## 3.3 The problem domain, substrate topology, and default geometric representation

In the experiments in this paper, HyperNEAT evolves controllers for a simulated quadruped. The default setup is the same from Clune et al. [1]. The ANN substrate consists of three two-dimensional, 5x4 Cartesian grids forming an input, hidden, and output layer (Fig. 2). There are no recurrent connections. Preliminary tests revealed that a hidden layer aided performance on this task, but recurrence did not. All possible connections between adjacent layers exist (although weights can be zero, functionally eliminating the link) meaning that there are $(5 \times 4)^2 \times 2 = 800$ links in each substrate. As in previous studies [1, 2, 4, 14], to facilitate the elimination of links, any link weight with a value less than 0.2 or greater than -0.2 is set to 0. Otherwise, the value is normalized to a range of -3 to 3.

The inputs to the substrate are the current angles of each of the 12 joints of the robot (described below), a touch sensor that provides a 1 if the lower leg is touching the ground and a 0 if it is not, the pitch, roll, and yaw of the torso, and a modified sine wave (which facilitated the production of periodic behaviors). The sine wave was the following function of time ($t$) in milliseconds: $\sin(120 \times t) \times \pi$. Multiplying by $\pi$ facilitates the production of numbers from $-\pi$ to $\pi$, which is the range of the unconstrained joints. The constant 120 was experimentally found to produce fast, natural gaits during preliminary tests, which also determined that the touch, pitch, roll, yaw, and sine inputs all improved the ability to evolve fast gaits [1].

The outputs of the ANN were the desired angle for each joint. This value was fed into a PID controller that simulated a servo. The controller subtracts the current joint angle from the desired joint angle. This difference was then multiplied by a

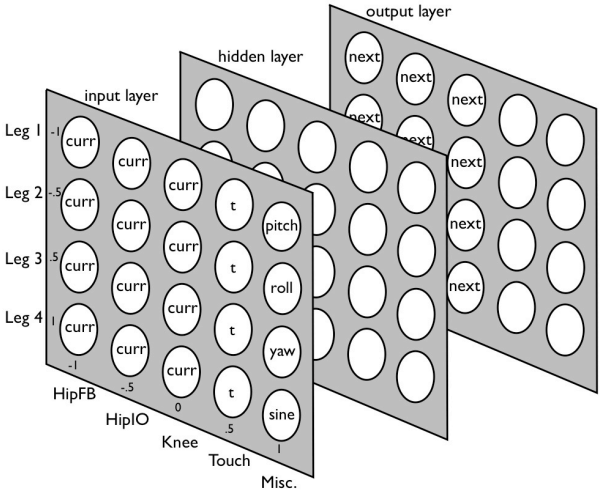


Fig. 2. The substrate configuration for HyperNEAT and FT-NEAT. The first four columns of each row of the input layer receive information about a single leg (the current angle of each of its three joints, and a 1 or 0 depending on whether the lower leg is touching the ground). The final column provides the pitch, roll, and yaw of the torso, as well as a sine wave. Evolution determines the roles of the hidden layer nodes. The nodes in the first three columns of each of the rows in the output layer specify the desired new joint angle. The joints will move toward that desired angle in the next time step, as described in the text. The outputs of the nodes in the rightmost two columns of the output layer are ignored.

constant (2.0), and a force of that magnitude was applied to the joint such that it would move toward the desired angle. Such PID-based controllers have been shown to be effective [1, 7, 9, 16].

The parameter configurations for HyperNEAT and FT-NEAT are from [1], and are similar to previous settings [2, 14]. They can be found at http://devolab.msu.edu/SupportDocs/HyperNEAT-SensitivityToGeometry. 50 trials were conducted for each treatment. Unless otherwise specified, trials within a treatment differed only in their random number generator seed, which influenced stochastic events, such as mutations. Each trial lasted 1,000 generations and had a population size of 150, which is common for HyperNEAT experiments [14].

Robots were evaluated in the ODE physics simulator [17]. The rectangular torso of the robot is (in arbitrary ODE units) 0.15 wide, 0.3 long, and .05 tall (Fig. 4). The shorter side of the robot in the forefront of Fig. 4 is designated the robot's front. Each of four legs is composed of three cylinders (length 0.075, radius 0.02) and three hinge joints. The first cylinder functions as a hip bone. It is parallel to the proximal-distal axis of the torso and barely sticks out from it. The second cylinder is the upper leg and the last cylinder is the lower leg. There are two hip joints and one knee joint. The first hip joint (HipFB) allows the legs to swing forward and backward (anterior-posterior) and is constrained to 180° such that, at maximum extension, it is parallel with the torso. The second hip joint (HipIO) allows a leg to swing in and out (proximal-distal). Together, the two hip joints approximate a universal joint. The knee joint swings forward and backward. The HipIO and knee joints are unconstrained.

Each controller was simulated for 6,000 time steps. Trials were cut short if any part of the robot except its lower leg touched the ground or if the number of direction changes in joints exceeded 960. The latter condition roughly reflects the physical fact that servo motors cannot be vibrated incessantly without breaking. The fitness of controllers was the following function of the maximum distance traveled in the X and Y dimensions: $2^{(X^2+Y^2)}$. The exponential nature of the function magnified the selective advantage of small increases in the distance traveled.

# 4. RESULTS and DISCUSSION

## 4.1 Engineered vs. Random Configurations

A test of the importance of choosing an appropriate geometric representation is to compare a human-engineered representation (Fig. 2) [1] to randomized representations. Such random configurations represent configurations created without intuitions about how to represent the geometric information of a problem, and could be produced by a naïve engineer or algorithm. Each random configuration has the geometric locations of the inputs and outputs scrambled within their layer. For each trial, the geometric representation was randomized at the beginning of a trial and remained unchanged throughout the trial. For example, the sine input, which is located at X=5, Y=4 in the engineered treatment (Fig. 2), may be at (1,1) in one randomized treatment and (3,2) in another. An average was calculated across 50 trials, each of which had a different randomized configuration.

The human-engineered configuration significantly outperformed the average of the random configurations (Fig. 3, $p < .05$, this and all future p values were generated with a Mann-Whitney U rank test). This performance difference shows that human intuitions about how to geometrically represent a problem help HyperNEAT. These results also underscore that the performance of HyperNEAT can be significantly affected by the geometric representation. It is also instructive to compare the randomized treatment to FT-NEAT. It has been previously shown that HyperNEAT (with the engineered configuration) outperforms FT-NEAT [1], so it is interesting to test whether a naïve geometric representation lowers the performance of HyperNEAT to the level of FT-NEAT. It turns out that HyperNEAT still performs better than FT-NEAT (Fig. 3), even with a randomized configuration ($p<.001$). This advantage could be due to HyperNEAT's generative ability to reuse link values, or to its ability to exploit geometric correlations that arise by chance in randomized configurations. Regardless of the reason, it is noteworthy that

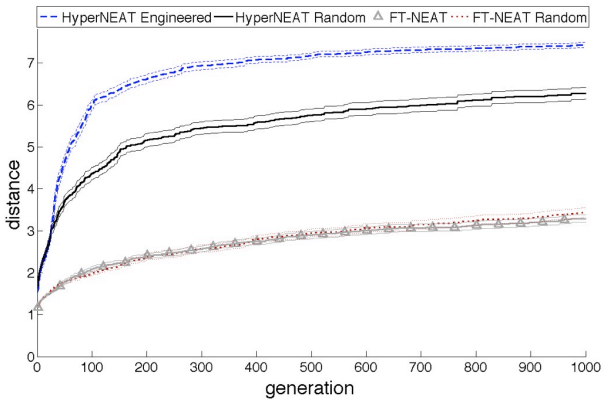


Fig. 3. The HyperNEAT default configuration vs. an average of randomized configurations and vs. direct encoding controls. Here, as in all figures, thick lines show averages and thin lines show one standard error of the mean.

HyperNEAT outperforms FT-NEAT even with a naïve representation of the geometric information. Unsurprisingly, FT-NEAT is not affected by a randomization of the geometry, which for FT-NEAT only changes the ordering of the values in its encoding ($p > .05$, Fig. 3).

## 4.2 Representations in Different Dimensions

To date, HyperNEAT has not been tested on the same problem with geometric representations in different *dimensions*. It is an open question as to whether the problem may be easier or harder for the CPPN to solve as the dimensionality of the problem representation more closely approximates the true geometry of the problem, which is three-dimensional in this case. To test this, HyperNEAT was evaluated separately with a one-dimensional (1-d), two-dimensional (2-d), and three-dimensional (3-d) representation. The 1-d treatment has only X coordinates, the 2-d treatment has only X and Y coordinates, and the 3-d treatment has X, Y, and Z coordinates. The coordinate values and geometric layout for each of these three treatments are shown in Fig. 4 (1-d and 3-d) and Fig. 2 (2-d). The number of CPPN inputs for each dimension is twice the number of dimensions, plus one for a bias. There are thus 3, 5 and 7 inputs to the CPPN, respectively, for the 1-d, 2-d, and 3-d treatments.

Interestingly, the 1-d representation performed significantly better than the 2-d and 3-d representations in the initial generations ($p < .05$ for generations 1-58, Fig. 5), but the 2-d and 3-d representations soon surpassed it ($p < .05$ for generations 170 on). It is possible that the 1-d representation is simpler, but less powerful, making it easier to learn, but harder to achieve high performance with. More tests are needed to reveal whether this

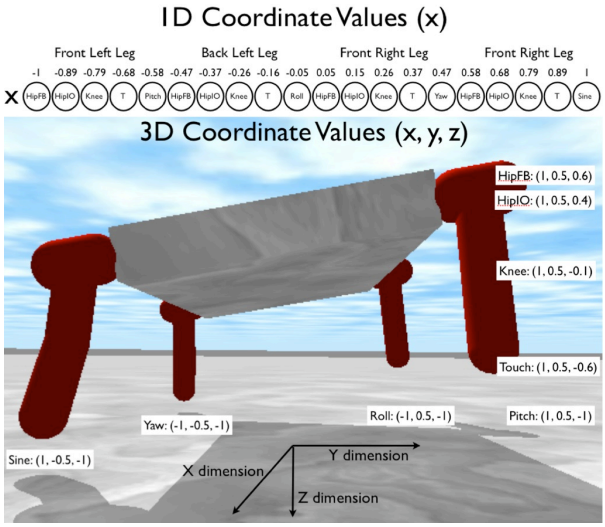


Fig. 4. The 1-d and 3-d geometric representations. For ease of viewing, only the input layer node coordinates are depicted. The numbering system is the same for hidden and output layers. The numbers shown are those fed to the CPPN when the corresponding node is the source node (or target node, for hidden or output layers) to determine the link weight between a source and target node. In the 3-d configuration, for three of the legs, only the roll, yaw, or sine node has its respective `X`, `Y`, and `Z` coordinates shown. The `X` and `Y` coordinates for the other nodes in each of those three legs will be the same as for the node shown for that leg, but the `Z` coordinate will change in the same manner as for the leg with all nodes shown.

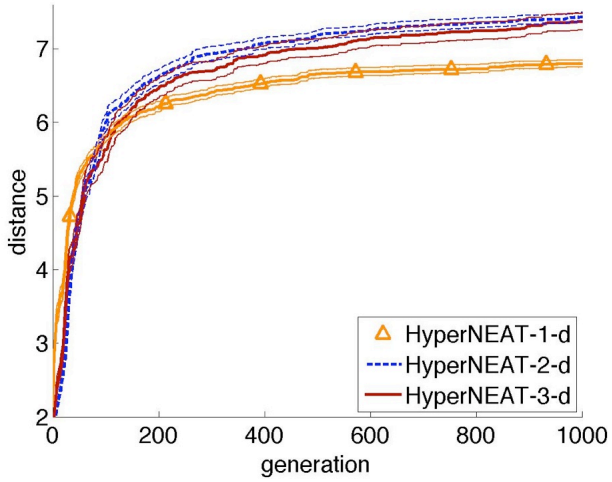


Fig. 5. The performance of representations in different dimensions.

phenomenon is general to HyperNEAT on most problems, or is specific to this domain. It could be the case that the 1-d representation was hampered because it is not very accurate with respect to the actual geometry of the robot problem. For example, it is difficult to represent all of the symmetries and repetitions of the robot in 1-d.

While the 2-d representation captures more of the geometric layout of the robot than the 1-d representation, it still lacks fidelity. On the robot, the two hip joints are in the same location and the knee is further away. However, the distance between these three joints is the same in the 2-d representation. Furthermore, the 2-d representation inaccurately represents the torso as a square instead of a rectangle. Finally, the 2-d representation does not represent both the front-back and left-right symmetry of the robot. While these issues could be creatively rectified in 2-d, they disappear when providing the true dimensions of the robot in 3-d.

Even in 3-d there remain some arbitrary choices when assigning geometric coordinates to inputs and outputs. Initially, even though the two hip joints occur in the same place on the actual robot, the CPPN would be unable to distinguish them if they had the exact same geometric coordinates. This problem was avoided by slightly separating these two joints in the representational geometry: The HipIO joint was placed just below the HipFB joint in the Z dimension. Additionally, the geometric coordinates must be determined for some information that has no meaningful geometric location. For example, where should the sine wave input go? This type of issue will always arise with HyperNEAT when dealing with information that does not belong to any geometric coordinate. The case of the pitch, roll, and yaw sensors is a bit clearer. It would be intuitive to place them in the torso, because that is what they provide feedback about, but it is not clear where in the torso they should be placed. For the purposes of this paper, placing the pitch, roll, and yaw sensors on the torso would have made a comparison with the 2-d configuration less clean, since in the 2-d setup the pitch, roll, yaw and sine (PRYS) inputs were placed just after the touch sensor on each leg. For this reason, the PRYS inputs were kept at the distal end of each leg.

The data reveal that the 2-d and 3-d treatments performed similarly throughout the experiment, and ended up statistically indistinguishable ($p > .05$, Fig. 5). Evidently, moving to a more accurate representation did not improve performance, which is consistent with a previous finding [3]. However, it is also interesting that the 3-d representation did not hurt HyperNEAT's performance. This result suggests that a user can select either a 2-d or 3-d setup depending on which is easier to implement. It is premature to extrapolate from one test in one problem domain, however, so more tests are needed to test the generality of these findings.

Comparing the engineered performance in each dimension against a randomized configuration from that dimension increases the sample size of the comparison between engineered and random configurations from 1 to 3 (although all three samples are from the same problem domain). The results, portrayed in Fig. 6, are relatively consistent across dimensions. In all cases, the engineered configuration outperformed the random configurations ($p < .001$) and the random configurations outperformed FT-NEAT ($p < .001$). Human intuitions provided a performance boost over the random treatments of 18.6%, 18.3%, and 11.7%, respectively, for the 1-d, 2-d, and 3-d representations.

It is also noteworthy that the 2-d-randomized treatment statistically ties the 3-d-randomized treatment ($p > .05$), meaning that the CPPN does no better or worse in either treatment due to the specific set of values fed to the CPPN (shown in Fig. 2 and Fig. 4). However, the 1-d-randomized treatment is significantly worse ($p < .01$) compared to both the randomized 2-d and randomized 3-d treatments. This result implies that one potential explanation for why the 1-d treatment did worse than the 2-d and 3-d treatments is because the CPPN has a harder time with the input numbers in the 1-d treatment (shown in Fig. 4), and not because the 1-d treatment is less geometrically accurate. The 1-d inputs could be harder to work with because of the specific numbers in that set, or because the numbers are close together, which could make it difficult to differentiate between them.

The data in Fig. 6 reveal that HyperNEAT outperforms FT-NEAT, even with a naïve, randomly chosen geometric configuration, regardless of the dimension of the representation. It is unknown to what extent the performance difference is due to HyperNEAT's generative capabilities or to its ability to exploit even randomized geometries. Unfortunately, these two forces are intertwined and difficult to experimentally isolate.

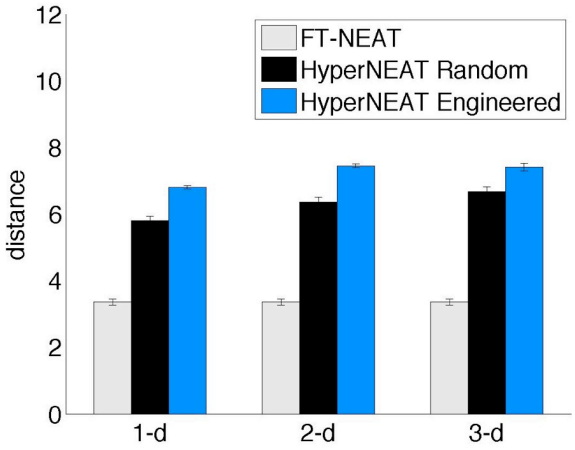


Fig. 6. Comparing the performances of an engineered configuration, random configurations and FT-NEAT in different dimensions.

## 4.3 Repeatedly Testing Random Representations

In the previous experiments, the randomized treatments featured one trial for each of 50 randomized configurations. The average across these configurations was worse than the engineered configuration and better than FT-NEAT. However, in the 1-d treatment, one of the randomized configuration trials outperformed all of the 1-d engineered trials and another randomized trial performed worse than many of the FT-NEAT trials. The variance in the results highlights the need to explore whether these configurations are inherently better or worse, or whether it was simply a stochastic idiosyncrasy during the individual trials that caused their extreme performance. To test whether certain randomized configurations might perform better than the engineered configuration or worse than FT-NEAT, 50 trials were conducted for the random configurations that performed best and worst, as well as for 25 other randomly chosen randomized configurations from the earlier experiment (Fig. 7).

Averages across 50 trials for the configurations that originally performed the best and the worst were not as extreme as the original individual trial scores produced by them. The idiosyncrasies of those single trials mattered more than any property of the configuration itself. That said, there is a substantial amount of variation between the 27 configurations tested, once again underscoring the effect that the geometric representation can have in HyperNEAT. The difference can be significant ($p < .001$ comparing the highest and lowest performing configurations). All of the variation in random configurations, however, was confined between the performance of FT-NEAT and HyperNEAT ($p < .01$). These data strongly recommend the selection of HyperNEAT over FT-NEAT on this problem. HyperNEAT's advantage may be because this problem is highly regular, since all legs can be correlated [1], and HyperNEAT increasingly outperforms FT-NEAT as problem-regularity increases [2]. It seems difficult, and may be impossible, to produce a geometric representation that performs worse than FT-NEAT. This result is surprising because

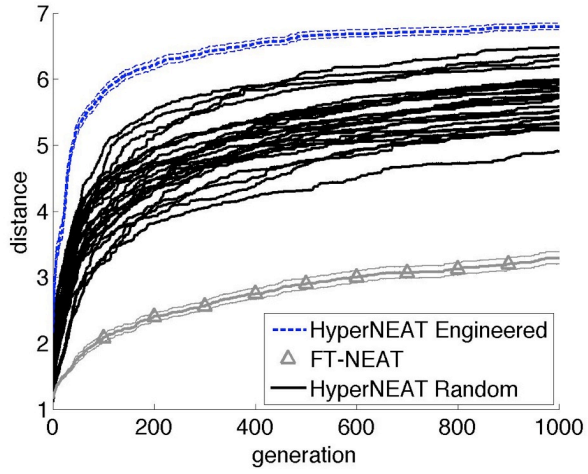


Fig. 7. A comparison of HyperNEAT 1-d and FT-NEAT to 27 randomized 1-d configurations. Each line is an average of 50 trials. Standard error bars are not shown for randomized configurations.

it suggests that HyperNEAT can exploit regularities in any geometric representation. As such, HyperNEAT could outperform its direct encoding alternatives even if a problem has no obvious geometry (provided the problem is regular [2]). It should be noted, however, that the direct encoding control was chosen because its ANN topology is identical to HyperNEAT. Additional studies (currently underway) will reveal whether other direct encodings, such as NEAT, better compete with HyperNEAT on this problem.

No random representation outperformed an engineered representation (Fig. 7). This outcome reinforces the fact that human intuitions about the geometry of a problem help us choose a rare subset of the possible space of geometric configurations that are high performing. Clearly, if the number of samples were increased, then, eventually, representations would be found that are equal to, and possibly better than, the engineered approach. However, those configurations may represent a tiny region of the search space that is hard to algorithmically find. Human engineers can easily select high-performing representations because of our intuitive grasp of geometry. It would be interesting in future work to evolve the geometric locations of nodes and compare the results to human-designed configurations.

## 4.4 Comparing Alternate Engineered Representations

In addition to comparing one engineered configuration to random configurations, it is illuminating to compare different engineered representations. Such tests are worthwhile because when arranging a configuration, some choices are difficult (because there seem to be many good alternatives) and others are arbitrary (because multiple options are seemingly equivalent). Whether it is important to investigate alternatives in both cases is addressed by comparing alternate 2-d configurations. For example, the location of the PRYS information is an arbitrary decision because, unlike the joints in each leg, the PRYS information does not have any obvious geometric location. The engineered solution places the PRYS information in the final column of a 5x4 substrate (Fig. 2). However, it could also have been placed as an additional row in a 4x5 substrate (hereafter referred to as the 'PRYS as row' setup).

Ideally, such arbitrary configuration details should not affect the CPPN. If it is the case that arbitrary decisions have little impact on evolution, then the designer does not need to spend time testing alternate configurations to find a better one. Unfortunately, the data shows that such configurations can make a difference (Fig. 8). The 'PRYS as row' treatment does 9.7% worse than the default setup, which is statistically significant ($p < .001$). While it would be interesting to test additional configurations (e.g., PRYS as the first column, or as the first row), limited computational resources prevented such investigations.

Other configuration decisions within a dimension may *a priori* be expected to have a larger impact. For example, the ordering of the legs may substantially affect the quality and type of gaits evolved. If CPPNs have an easier time grouping nodes that are closer to each other, then placing certain legs next to each other in the Y dimension in the 2-d setup may make it more likely for those legs to have similar neural controllers and hence have coordinated movements. Thus, some leg orderings may be more likely to produce left-right symmetry than front-back symmetry, for instance, which could affect fitness scores if one type of symmetry tends to produce faster gaits.

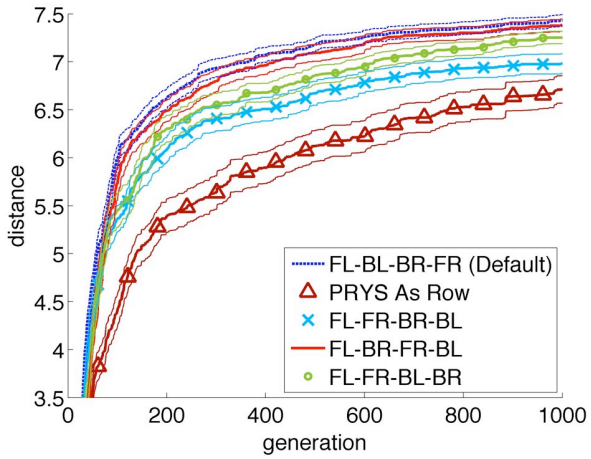


Fig. 8. The performance of alternate 2-d engineered configurations.

Three alternate orderings were tested in addition to the default ordering (Fig. 8). Experiments with these configurations (Table 1) reveal that the default setup (FL-BL-BR-FR) is statistically indistinguishable from the FL-BR-FR-BL ordering ($p > .05$). However, the other two leg orderings (FL-FR-BR-BL and FL-FR-BL-BR) performed worse than the default ($p < .01$). These data suggest that evolution did worse when front-back symmetry was encouraged (by ordering the legs F*F*B*B*, where * is a wildcard). A more exhaustive test of different configurations is warranted, but was prevented by limited computational resources. Nevertheless, these results do conclusively show that the order in which the legs are numbered for the CPPN can make a difference.

Another way of investigating the effect of different leg orderings is to classify the gaits produced by each representation. The gait of the best controller from all 50 trials in each of the four treatments was viewed and categorized (Table 1). In all of the treatments, the overwhelming majority of gaits moved all four legs in synchrony. However, the exceptions to this rule within each treatment are interesting because they reflect the geometric biases of each configuration. For example, all four cases of left-right symmetry evolved in the configuration that ordered the legs *L*L*R*R. Furthermore, all seven cases of front-back symmetry were seen in the only two configurations that placed the legs in the order F*F*B*B*. It seems that it is much easier for the CPPN to bisect the Y dimension than to group legs 0 and 2 into one group and 1 and 3 into another. This 'every other' grouping requires a more complicated function, and did not evolve in any of the best controllers. Interestingly, the configuration chosen to encourage a trot gait, where diagonal legs are in sync (FL-BR-FR-BL), evolved neither a diagonally-symmetric gait nor a gait with front-back or left-right symmetry. For some reason, possibly because the torso is inflexible, the trot gait was not employed by evolution. That, plus the difficulty of grouping the left-right legs or the front-back legs in this configuration, is probably the reason that no diagonal, left-right or front-back symmetries evolved.

Further evidence of the influence of the geometric configuration on the resultant gait can be seen by examining those gaits in which three of the legs moved in synchrony, and one leg did something different. In 23 out of 25 (92%) of these gaits, the exception leg was the last leg in the ordering. It is not surprising that it is easier for the CPPN to make one distinction (e.g., all legs

**Table 1: Resultant gait types for different leg orderings. Gaits were placed into the following categories. 4way Sym(metry): all legs in synchrony, L-R Sym: the left legs are in phase, and the right legs out of phase, F-B Sym: the front legs are in phase, and the back legs are out of phase, One Leg Out of Phase: three legs moved in synchrony and one was out of phase (resembles a gallop). If two legs were motionless, they were considered to be in synchrony. 2 gaits did not fit these categories, and are not tabulated. FL=Front Left, BL=Back Left, BR=Back Right and FR=Front Right.**

| | 4way Sym | L-R Sym | F-B Sym | One Leg Out Of Phase | | | |
|---|---|---|---|---|---|---|---|
| | | | | FL | BL | BR | FR |
| FL-BL-BR-FR (default) | 36 | 4 | | | | | 9 |
| FL-BR-FR-BL | 47 | | | | 2 | | 1 |
| FL-FR-BL-BR | 44 | | 3 | | | 1 | |
| FL-FR-BR-BL | 36 | | 4 | | 9 | | 1 |

less than N) instead of the two distinctions that are required to pick a leg out of the middle of a dimension. The two exceptions, however, prove that it is possible for the CPPN to make an exception for a middle leg. It is not clear why the CPPN tended to single out the last leg and not the first.

It appears that the ordering of components geometrically can bias HyperNEAT's grouping of those components. This result, which has not been previously reported, means that a user can inject biases (desired or not) into how HyperNEAT clusters subcomponents of a problem. For example, if evolving a team of multiple agents, which HyperNEAT has been shown to do well [3], the geometric ordering could influence the types of teams selected. If the ordering were speedster-speedster-tank-tank, for example, then the result may be more likely to involve a speedster squadron and a tank squadron. A speedster-tank-speedster-tank ordering, on the other hand, may be more likely to produce two heterogeneous speedster-tank teams. Importantly, the bias of any configuration is also determined by the CPPN function set, and changing it could alter the biases of any given representation.

# 5. CONCLUSION

This paper shows that when evolving controllers for simulated legged robots, HyperNEAT can be sensitive to the way its geometric information is represented. HyperNEAT outperformed a direct encoding control even with randomized geometric representations. HyperNEAT's success with random configurations suggests it can perform well even if one does not know how to geometrically represent a problem. However, properly choosing a geometric configuration, which may seem intuitive to a human engineer, can provide a performance increase (10%-20% on this problem). Testing alternate engineered configurations was shown to be important for two reasons: Initially, some seemingly arbitrary decisions in the design of geometric representations can have large effects. Additionally, alternate options that *a priori* seem good for different reasons can have significantly different performance levels. In addition to quantitative fitness effects, the geometric configuration can also affect the *types* of solutions evolved, enabling engineers to bias the products of HyperNEAT evolution. HyperNEAT's sensitivity to its geometric representation is both detrimental, because work is required to optimize it, and powerful, because altering it can yield performance increases and enable engineers to shape the solutions produced. It is important to note, however, that all of the conclusions in this paper are drawn from one problem domain. Future work is required to see whether such conclusions hold more generally.

# 6. ACKNOWLEDGMENTS

Thanks to NSF grant # CCF-0643952, the Templeton Foundation, Ken Stanley, Benjamin Beckmann, and the reviewers.

# 7. REFERENCES

[1] J. Clune, B. E. Beckmann, C. Ofria, and R. T. Pennock, "Evolving coordinated quadruped gaits with the HyperNEAT generative encoding." Proc. IEEE Congress on Evolutionary Computing Special Section on Evolutionary Robotics, 2009. Trondheim, Norway.

[2] J. Clune, C. Ofria, and R. T. Pennock, "How a generative encoding fares as problem-regularity decreases," in PPSN (G. Rudolph, T. Jansen, S. M. Lucas, C. Poloni, and N. Beume, eds.), vol. 5199 of Lecture Notes in Computer Science, pp. 358–367, Springer, 2008.

[3] D. B. D'Ambrosio and K. O. Stanley, "Generative encoding for multiagent learning," in GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation, (New York, NY, USA), pp. 819–826, ACM, 2008.

[4] J. Gauci and K. O. Stanley, "A case study on the critical role of geometric regularity in machine learning," in AAAI (D. Fox and C. P. Gomes, eds.), pp. 628–633, AAAI Press, 2008.

[5] F. Gruau, "Automatic definition of modular neural networks," Adaptive Behaviour, vol. 3, no. 2, pp. 151–183, 1995.

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

[7] G. Hornby, S. Takamura, T. Tamamoto, and M. Fujita, "Autonomous evolution of dynamic gaits with two quadruped robots," IEEE Transactions on Robotics, vol. 21, no. 3, pp. 402–410, 2005.

[8] J. Kodjabachian and J-A Meyer, "Evolution and Development of Neural Controllers for Locomotion, Gradient-Followig, and Obstacle-Avoidance in Artificial Insects." IEEE Transactions on Neural Networks. Vol 9:5, pp. 796-812. Sept. 1998.

[9] H. Liu and H. Iba, "A hierarchical approach for adaptive humanoid robot control," in Proceedings of the 2004 IEEE Congress on Evolutionary Computation, (Portland, Oregon), pp. 1546–1553, IEEE Press, 20-23 June 2004.

[10] S. Nolfi and D. Floreano, Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines. Bradford Book, 2004

[11] K. Sims, "Evolving 3d morphology and behavior by competition," Artif. Life, vol. 1, no. 4, pp. 353–372, 1994.

[12] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," Evol. Comput., vol. 10(2), no. 2, pp. 99–127, 2002.

[13] K. O. Stanley, "Compositional pattern producing networks: A novel abstraction of development," Genetic Programming and Evolvable Machines, vol. 8, pp. 131 – 162, June 2007.

[14] 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). 2009. To be published.

[15] V. K. Valsalam and R. Miikkulainen, "Modular neuroevolution for multilegged locomotion," in GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation, (New York, NY, USA), pp. 265–272, ACM, 2008.

[16] K. Wolff and P. Nordin, "An evolutionary based approach for control programming of humanoids," in Proceedings of the 3rd International Conference on Humanoid Robots (Humanoids'03), (Karlsruhe, Germany), IEEE, VDI/VDE-GMA, 1-2 October 2003.

[17] www.ode.org

[18] www.picbreeder.org