On the Performance of Indirect Encoding Across the Continuum of Regularity

Jeff Clune, Kenneth O. Stanley, Robert T. Pennock, and Charles Ofria

***Abstract*—This paper investigates how an evolutionary algorithm with an indirect encoding exploits the property of phenotypic regularity, an important design principle found in natural organisms and engineered designs. We present the first comprehensive study showing that such phenotypic regularity enables an indirect encoding to outperform direct encoding controls as problem regularity increases. Such an ability to produce regular solutions that can exploit the regularity of problems is an important prerequisite if evolutionary algorithms are to scale to high-dimensional real-world problems, which typically contain many regularities, both known and unrecognized. The indirect encoding in this case study is HyperNEAT, which evolves artificial neural networks (ANNs) in a manner inspired by concepts from biological development. We demonstrate that, in contrast to two direct encoding controls, HyperNEAT produces both regular behaviors and regular ANNs, which enables HyperNEAT to significantly outperform the direct encodings as regularity increases in three problem domains. We also show that the *types* of regularities HyperNEAT produces can be biased, allowing domain knowledge and preferences to be injected into the search. Finally, we examine the downside of a bias toward regularity. Even when a solution is mainly regular, some irregularity may be needed to perfect its functionality. This insight is illustrated by a new algorithm called HybrID that hybridizes indirect and direct encodings, which matched HyperNEAT's performance on regular problems yet outperformed it on problems with some irregularity. HybrID's ability to improve upon the performance of HyperNEAT raises the question of whether indirect encodings may ultimately excel not as stand-alone algorithms, but by being hybridized with a further process of refinement, wherein the indirect encoding produces patterns that exploit problem regularity and the refining process modifies that pattern to capture irregularities. This paper thus paints a more complete picture of indirect encodings than prior studies because it analyzes the impact of the continuum between irregularity and regularity on the performance of such encodings, and ultimately suggests a path forward that combines indirect encodings with a separate process of refinement.**

***Index Terms*—Artificial neural networks, indirect encodings, generative encodings, developmental encodings, HyperNEAT, regularity.**

Manuscript received March 31, 2010. This work was supported in part by NSF grants CCF-0643952, CCF-0750787, CNS-0751155, CCF-0820220, CNS-0915855, and DBI-0939454, the Cambridge Templeton Consortium, U.S. Army Grant W911NF-08-1-0495, a Quality Fund Grant from MSU, the DARPA FunBio program, and an NSF Postdoctoral Research Fellowship in Biology to JC (award number DBI-1003220). Jeff Clune (jeffclune@cornell.edu, 517.214.1060) is in the Department of Mechanical and Aerospace Engineering at Cornell University in Ithaca, NY, USA. Kenneth O. Stanley is in the School of Electrical Engineering and Computer Science at the University of Central Florida in Orlando, FL, USA. Robert T. Pennock (pennock5@msu.edu) and Charles Ofria (ofria@msu.edu) are in the BEACON Center for the Study of Evolution in Action and the Department of Computer Science and Engineering at Michigan State University (MSU) in East Lansing, MI, USA. Robert T. Pennock is also in the Department of Philosophy and the Lyman Briggs College at MSU.

I. Introduction and Motivation

A Long-standing challenge for those who work with evolutionary algorithms (EAs) is to synthetically evolve entities that rival or surpass the complexity of both natural organisms and designs produced through human engineering. A key design principle critical to the success of both is regularity. *Regularity* refers to the compressibility of the information describing a structure, and typically involves symmetries and the repetition of modules or other design motifs [28]. Regularities allow solutions to sub-problems to be reused in a design, as in the cells of a body or the wheels of a car.

The level of regularity that an EA tends to produce is affected by the *encoding*, which is how information is stored in the genome and the process by which that information produces the phenotype. Regularity in evolved solutions is less likely to emerge with *direct encodings*, wherein each element in the genotype encodes an independent aspect of the phenotype. In contrast, regularities are common with *indirect encodings*, wherein information in the genome can be *reused* to affect many parts of the phenotype [46]. Natural organisms are regular largely because they are encoded indirectly [2], [3], [32].

Reusing genetic information also facilitates scalability. With indirect encodings, evolution can search in a low-dimensional space yet produce phenotypes with many more dimensions. For example, only about 25000 genes encode the information that produces the trillions of cells that make up a human [42].

Many prior studies have shown that indirect encodings often outperform direct encodings [10], [15], [20], [21], [23], [30], [34], [44], [46]. However, in each case the problem domain is highly regular, or the regularity of the problem is unspecified and ambiguous. To date, no systematic study has been performed on how indirect encodings perform across a continuum from regular problems to irregular problems. This gap in our knowledge raises the question of whether indirect encodings achieve their increased performance on regular problems at the expense of performing poorly on problems with intermediate or low regularity. To fill that gap, this paper provides a systematic comparison of an indirect encoding and two direct encoding controls on multiple problems as problem regularity is varied from high to low.

The indirect encoding in this paper is motivated by a key concept from developmental biology. Nature builds complex organisms by producing increasingly complex geometric coordinate frames, and then determining the fate of phenotypic elements as a function of their location within such frames [2]. This process produces phenotypes with regularity, modularity, and hierarchy [2], [48], which are beneficial design principles [28]. Yet the exploitation of geometric information in natural organisms is challenging because each phenotypic element (e.g., a cell) does not have access to its global geometric position. Developing organisms therefore first have to generate such positional information via chemical gradients before exploiting it. In synthetic evolution, however, we can skip this first step by assigning geometric coordinates to phenotypic elements and allowing a genome to determine the fates of phenotypic elements as a function of these coordinates [43]. This technique allows evolution to start with geometric information and immediately exploit it, instead of first needing to discover how to produce geometric information before exploiting it.

One encoding that harnesses this idea is Hypercube-based NeuroEvolution of Augmenting Topologies, or HyperNEAT. HyperNEAT's explicit incorporation of geometry enables it to exploit geometric relationships in a problem domain [8], [16], [17], [44]. HyperNEAT has performed well on a wide range of problems, such as generating gaits for legged robots [5], pattern recognition [44], controlling multi-agent teams [9], [11], and evaluating checkers boards [16], [17]. Additionally, on a soccer keepaway task that is a common benchmark for reinforcement learning algorithms, HyperNEAT produced the highest score to date for any type of algorithm [52]. Because HyperNEAT abstracts a key geometric ingredient that drives the success of natural development, and because empirically it has been shown to be a promising encoding, it is interesting to observe the regularities that HyperNEAT produces. Specific questions can be addressed, such as what types of regularities HyperNEAT generates (e.g., repeating, symmetric, etc.), whether it can create variations on repeated themes instead of being restricted to identical repetitions, and what kinds of irregular exceptions it can make to the patterns it generates. An additional question of interest is whether the experimenter can bias the regularities that HyperNEAT produces to inject domain knowledge into the algorithm. All of these questions are addressed in this paper.

Results in this paper show that HyperNEAT exploits even intermediate problem regularity, and thus increasingly outcompetes direct encoding controls as problem regularity increases. HyperNEAT achieves this success by producing regular artificial neural networks (ANNs) that in turn produce regular behaviors. HyperNEAT also proves more evolvable than direct encoding controls and its solutions generalize better.

However, an important accompanying result is that HyperNEAT's performance decreases on irregular problems, partly because of its bias toward producing regularities. To investigate this effect further, we introduce a new algorithm called HybrID that allows the HyperNEAT indirect encoding to produce regular patterns in concert with a direct encoding that can modify these patterns to produce irregularities. HybrID matches HyperNEAT's performance on regular problems, but outperforms HyperNEAT on problems with irregularity, which demonstrates that HyperNEAT struggles to generate certain kinds of irregularity on its own. The success of HybrID raises the interesting question of whether indirect encodings may truly excel not as stand-alone algorithms, but in combination with a further process that refines their regular patterns. Intriguingly, this further process in nature could be lifetime adaptation via learning.

In the following sections, we introduce HyperNEAT, two direct encoding controls, and the three problem domains. We then relate the experimental results and discuss them before offering concluding remarks.

II. HyperNEAT and the Direct Encoding Controls

In this section, we describe the indirect encoding HyperNEAT and two direct encodings that serve as controls.

A. HyperNEAT

In 2007 an encoding was introduced called Compositional Pattern Producing Networks (CPPNs), which abstracts the process of natural development without requiring the simulation of diffusing chemicals [43]. When CPPNs encode ANNs, the algorithm is called HyperNEAT [44], which is described in detail below. A key idea behind CPPNs is that complex patterns can be produced by determining attributes of their phenotypic components as a function of their geometric location. This idea is based on the belief that cells (or higher-level modules) in nature often differentiate into their possible types (e.g., kidney, liver, etc.) as a function of where they are situated in geometric space. For example, for some insects, a segment at the anterior pole should produce antennae and a segment at the posterior pole should produce a stinger.

Components of natural organisms cannot directly determine their location in space, so organisms have evolved developmental processes that create chemical gradients that organismal components use to figure out where they are and, thus, what to become [2]. For example, early in the development of embryos, different axes (e.g., anterior-posterior) are indicated by chemical gradients. Additional gradients signaled by different proteins can exist in the same area to represent a different pattern, such as a repeating motif. Downstream genes, such as Hox genes, can then combine repeated and asymmetric information to govern segmental differentiation [2]. Further coordinate frames can then be set up within segments to govern intra-module patterns.

1) CPPNs: One of the key insights behind CPPNs is that cells *in silico* can be directly given their geometric coordinates. The CPPN genome is a function that takes geometric coordinates as inputs and outputs the fate of an organismal component. When CPPNs encode two-dimensional pictures, the coordinates of each pixel on the canvas (e.g., $x = 2$, $y = 4$) are iteratively passed to the CPPN genome, and the output of the function is the color or shade of the pixel (Fig. 1).

Each CPPN is a directed graph in which every node is itself a single function, such as sine or Gaussian. The nature of the functions can create a wide variety of desirable properties, such as symmetry (e.g., a Gaussian function) and repetition (e.g., a sine function) that evolution can exploit. Because the genome allows functions to be made of other functions, coordinate frames can be combined and hierarchies can develop. For instance, a sine function early in the network

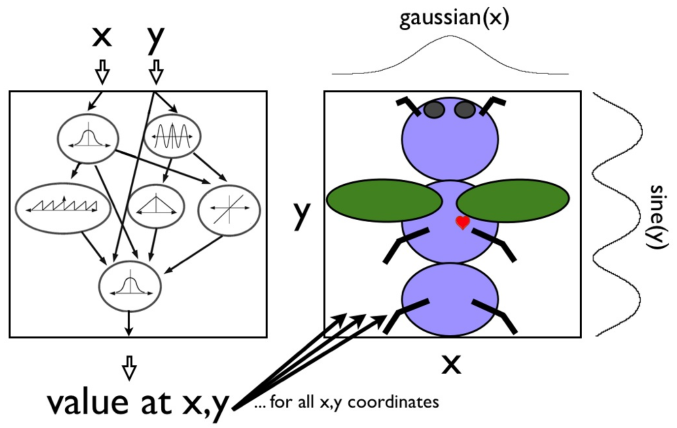

Fig. 1. Compositional Pattern Producing Networks. CPPNs compose mathematical functions to generate regularities, such as symmetries and repeated modules, with and without variation. This figure is adapted from Stanley (2007).

can create a repeating theme that, when passed into the symmetrical Gaussian function, creates a repeating series of symmetrical motifs (Fig. 1). This procedure is similar to the natural developmental processes described above [2].

The links that connect and allow information to flow between nodes in a CPPN have a weight that can magnify or diminish the values that pass along them. Mutations that change these weights may, for example, give a stronger influence to a symmetry-generating part of a network while diminishing the contribution from another part.

When CPPNs are evolved artificially with humans performing the selection, the evolved shapes look complex and natural (Fig. 2) [38]. Moreover, these images display the features in natural organisms that indirect encodings were designed to produce, namely, symmetries and the repetition of themes, with and without variation.

2) Encoding ANNs with CPPNs: In the HyperNEAT algorithm, CPPNs encode ANNs instead of pictures, and evolution modifies the population of CPPNs [17], [44]. HyperNEAT evolves the weighs for ANNs with a fixed topology. The ANNs in the experiments in this paper feature a two-dimensional, $m \times n$ Cartesian grid of inputs and a corresponding $m \times n$ grid of outputs. If an experiment uses an ANN with hidden nodes, the hidden nodes are placed in their own two-dimensional layer between the input and output grids. Recurrence is disabled, so each of the $m \times n$ nodes in a layer has a link of a given weight to each of the $m \times n$ nodes in the proximate layer, excepting output nodes, which have no outgoing connections. Link weights can be zero, functionally eliminating a link.

The inputs to the CPPNs are a constant bias value and the coordinates of both a source node (e.g., $x_1 = 0, y_1 = 0$) and a target node (e.g., $x_2 = 1, y_2 = 1$) (Fig. 3). The CPPN takes these five values as inputs and produces one or two output values, depending on the ANN topology. If there is no hidden layer in the ANN, the single output is the weight of the link between a source node on the input layer and a target node on the output layer. If there is a hidden layer, the first output value determines the weight of the link between the associated input (source) node and hidden-layer (target) node, and the second output value determines the weight of the link between the associated hidden-layer (source) node and output-layer (target) node. All pairwise combinations of source and target node coordinates are iteratively passed as inputs to the CPPN to determine the weight of each ANN link. HyperNEAT can thus produce patterns in weight space similar to the patterns it produces in two-dimensional pictures (Fig. 2).

An additional novel aspect of HyperNEAT is that it is one of the first neuroevolutionary algorithms capable of exploiting the geometry of a problem [8], [16], [17], [44]. Because the connection weights between nodes 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, HyperNEAT can exploit such information. For example, when playing checkers, the concept of adjacency (on the diagonals) is important. Connection weights between neighboring squares may need to be different than weights between distant squares. HyperNEAT can create this kind of connectivity motif and repeat it across the board [16], [17].

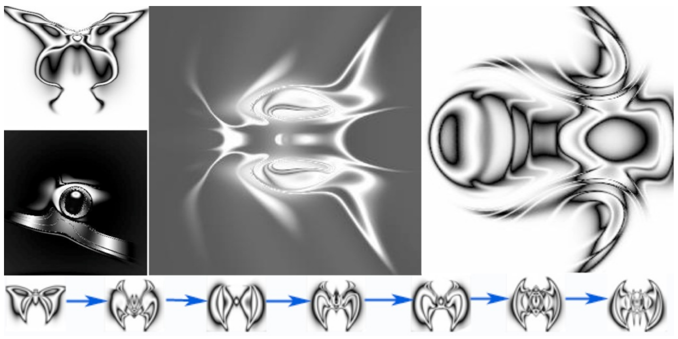

Fig. 2. Images Evolved with CPPNs. Displayed are pictures from picbreeder.org [38], a website where visitors select images from a population evolved with the CPPN indirect encoding, which is also used in HyperNEAT. The bottom row shows images from a single lineage. Arrows represent intermediate forms that are not pictured.

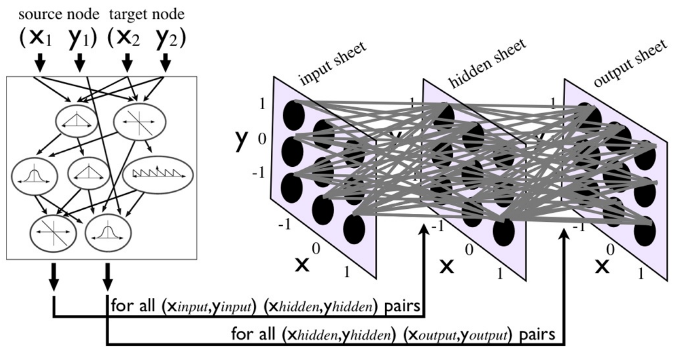
source node target node
(x_1 y_1) (x_2 y_2)
input sheet
hidden sheet
output sheet
y
1
0
-1
-1
x
0
1
for all (x_{input},y_{input}) (x_{hidden},y_{hidden}) pairs
for all (x_{hidden},y_{hidden}) (x_{output},y_{output}) pairs

Fig. 3. HyperNEAT Produces ANNs from CPPNs. Weights are specified as a function of the geometric coordinates of the source node and the target node for each connection. The coordinates of these nodes and a constant bias are iteratively passed to the CPPN to determine each connection weight. If there is no hidden layer, the CPPN has only one output, which specifies the weight between the source node in the input layer and the target node in the output layer. If there is a hidden layer in the ANN, the CPPN has two output values, which specify the weights for each connection layer as shown. This figure is adapted from Gauci and Stanley (2008).

Producing such a regularity would be more difficult with an encoding that does not include geometric information, because there would be no easy way for such an algorithm to identify which nodes are adjacent.

Variation in HyperNEAT occurs when mutations or crossover change the CPPNs. Mutations can add a node, which results in the addition of a function to the network, or change its link weights. The allowable functions for CPPNs in this paper are sine, sigmoid, Gaussian, and linear. The evolution of the population of CPPN networks in HyperNEAT occurs according to the principles of the widely used NeuroEvolution of Augmenting Topologies (NEAT) algorithm [45]. NEAT, which was originally designed to evolve ANNs, can be fruitfully applied to CPPNs because a population of CPPNs is similar in structure to a population of ANNs.

The NEAT algorithm contains three major components [45], [47]. First, 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. Second, NEAT uses a fitness-sharing mechanism that preserves diversity in the population and allows new innovations time to be tuned by evolution before forcing them to compete against rivals that have had more time to mature. Finally, historical information stored in genes helps to perform crossover in a way that is effective, yet avoids the need for expensive topological analysis. A full explanation of NEAT can be found in Stanley & Miikkulainen (2002).

B. FT-NEAT, a Direct Encoding Control for HyperNEAT

A common direct encoding control for HyperNEAT is *Fixed-Topology NEAT* (FT-NEAT, also called Perceptron NEAT or P-NEAT when it does not have hidden nodes) [4]–[8], [44]. FT-NEAT is similar to HyperNEAT in all ways, except it directly evolves each weight in the ANN independently instead of determining link weights via an indirect CPPN. In other words, for FT-NEAT there is no distinction between the genotype and phenotype, in contrast to HyperNEAT (where the CPPN genome network encodes a different ANN phenotype). All other elements from NEAT (e.g., its crossover and diversity preservation mechanisms) remain the same between HyperNEAT and FT-NEAT. Additionally, the number of nodes in the ANN phenotype are the same between HyperNEAT and FT-NEAT. Mutations in FT-NEAT cannot add nodes, making FT-NEAT a degenerate version of NEAT. Recall that the complexification in HyperNEAT is performed on the CPPN genome, but that the number of nodes in the resultant ANN is fixed. The end product of HyperNEAT and FT-NEAT are thus ANNs with the same number of nodes, whose weights are determined in different ways.

C. NEAT, a Second Direct Encoding Control for HyperNEAT

While FT-NEAT is a good control for HyperNEAT because it holds the number of nodes in the ANN constant, HyperNEAT should also be compared against a cutting-edge direct encoding neuroevolution algorithm, such as regular NEAT [45]. The only difference between FT-NEAT and NEAT is that in NEAT hidden nodes and connections can be added during evolution. In those experiments in this paper where the optimal number of hidden nodes is not known a priori, we compare HyperNEAT to NEAT in addition to FT-NEAT.

D. Parameter Settings

The parameters for all of the experiments below follow standard HyperNEAT and FT-NEAT conventions [4]–[8], [44] and can be found online at http://devolab.msu.edu/SupportDocs/Regularity. The results in this paper were found to be robust to moderate variations of these parameters.

III. The Three Experimental Problem Domains

The first two problem domains are *diagnostic problems* that are designed to determine how the algorithms perform as regularity is varied from low to high. The first problem, called *Target Weights*, enables regularity to be scaled from zero to complete. However, this problem has no interactions between the different problem components. The second problem, *Bit Mirroring*, adds such interactions between problem components, making it a challenging problem with different types of regularity that can each be adjusted, yet the optimal solution in each case is known. The third problem, called the *Quadruped Controller* problem, is a challenging, real-world problem in which the regularity can be scaled and the optimal solution is not known. While we have previously reported some results in these domains [5]–[8], this paper provides a more extensive investigation of the performance of HyperNEAT and direct encoding controls on these problems, including additional experiments, analyses, controls, and alternate versions of the problems.

A. The Target Weights Problem

One way to create a completely irregular problem is to challenge evolution to evolve an ANN phenotype (P) that is identical to a target neural network (T), where T is completely irregular. Regularity can then be scaled by increasing the regularity of T (Fig. 4). We call this the Target Weights problem because evolution is attempting to match a target vector of weights (recall that the number of nodes is constant, so the vector of weights in T fully describes T). Fitness is a function of the difference between each weight in P and the corresponding weight in T, summed across all weights. The lower this summed error is, the higher the fitness value. Specifically, the proximity to the target is calculated as

$$proximity\ to\ target = \sum_{i=1}^{N} M - |P_i - T_i|, \qquad (1)$$

where N is the number of weights, M is the maximum error possible per weight (which is 6 because weights could range from -3 to 3), P_i is the value of the ith weight in the phenotype, and T_i is the value of the ith weight in the Target ANN. To amplify the importance of small improvements, fitness is then calculated as

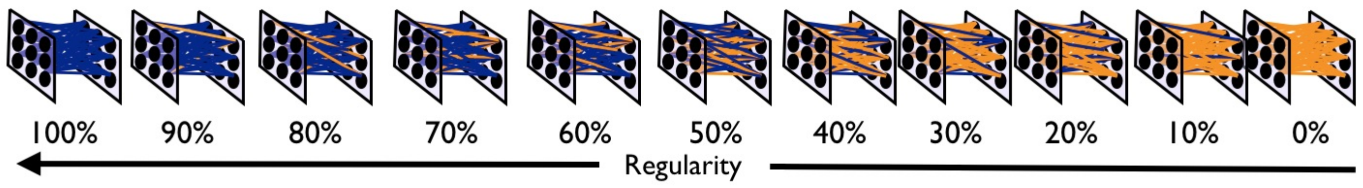

Fig. 4. Scaling Regularity in the Target Weights Problem. Regularity is scaled by changing the percentage of weights S (which increases from right to left) that are set to Q, a single randomly-chosen value (shown as dark blue lines). The remaining weights are each set to a random number (shown as light orange lines).

$$fitness = 2^{proximity\ to\ target}. \quad (2)$$

To scale the regularity of this problem, some randomly chosen subset of the target weight values, S, are assigned Q, a single randomly-chosen value. All remaining weight values are independently assigned a random value. Changing the number of weights in S scales the regularity of the problem. When S is set to 0%, all of the target weight values are chosen independently at random. When S is set to 50%, half the weights have the value Q and the rest have values independently chosen at random. In the most regular version of the problem, S is set to 100% and all weights have the value Q. There are 11 treatments, with S values of $0, 10, 20...100$, and 10 runs per treatment. Target vectors are constant for each evolutionary run, but are different between runs (due to differences in randomly-generated weight values, including Q). Trials last 1000 generations with a population size of 1000. The ANNs have 3×3 grids of input and output neurons. Because the number of nodes in this problem does not change, only FT-NEAT is tested as a direct encoding control.

The Target Weights problem is useful because it allows regularity to be scaled from zero to complete, and because the regularity of the solution is known *a priori*. It is also a simplistic problem because it has no interactions (epistasis) between weight elements (i.e., changing a given weight will not affect the optimal value of other weights). While this property makes it a good starting point for investigating regularity, other more epistatic problems are also required to understand performance in more realistic scenarios.

B. The Bit Mirroring Problem

The Bit Mirroring problem is intuitively easy to understand, yet provides multiple types of regularities, each of which can be scaled independently. For each input, a target output is assigned (e.g., the input $x_1 = -1, y_1 = -1$ could be paired with output $x_2 = 0, y_2 = 1$; Fig. 5a). A value of one or negative one is randomly provided to each input, and the fitness of an organism is incremented if that one or negative one is reflected in the target output. Outputs greater than zero are considered 1, and values less than or equal to zero are considered -1. The correct wiring is to create a positive weight between each input node and its target output and, importantly, to set to zero all weights between each input node and its non-target output nodes (Fig. 5a). To reduce the effect of randomness in the inputs, in every generation each organism is evaluated on ten different sets of random inputs and these

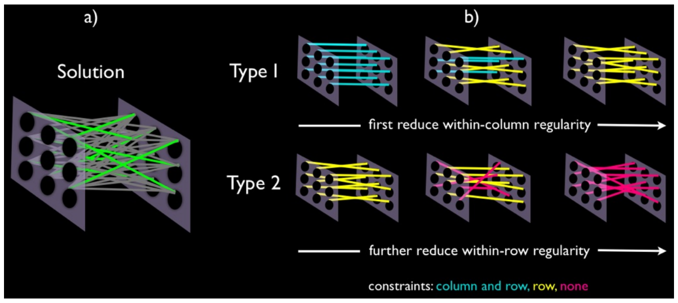
a)
b)
Solution
Type I
Type 2
first reduce within-column regularity
further reduce within-row regularity
constraints: column and row, row, none

Fig. 5. The Bit Mirroring Problem. **(a)** The correct wiring motif for the links projecting from each input node is to create an excitatory connection (light green) to the correct target output node, and to turn off all other links (dark gray). **(b)** Within-column regularity **(Type 1)** is highest when all targets are in the same column, and can be lowered by decreasing the number of targets in the same column (by assigning unconstrained targets to columns at random). For the experiments in this paper, within-column regularity is scaled while keeping within-row regularity at its highest possible level, with all targets in the same row. Within-row regularity **(Type 2)** is reduced by constraining fewer targets to be in the same row. By first reducing within-column regularity, then further reducing within-row regularity, the overall regularity of the Bit Mirroring problem can be smoothly scaled from high to low. Note that the treatment with the lowest Type 1 regularity and the highest Type 2 regularity have identical constraints.

scores are summed to produce the fitness for that organism. The max fitness is thus $10n^2$, where n is the number of nodes in the input sheet. Each run lasted 2000 generations and had a population size of 500. As in Target Weights, the number of nodes does not change on this problem (additional nodes would only hurt performance), so only FT-NEAT is tested as a control.

The first type of regularity in the problem is *within-column regularity*. This regularity is high when targets are in the same column, and low when there is no expectation about which column a target will be in (Type 1 in Fig. 5b). The second type of regularity is *within-row regularity*, which is the likelihood that a target is in the same row (Type 2 in Fig. 5b). While these two regularities are intuitively related, evolutionary algorithms must compute them independently, which means they are distinct. Each of these types of regularity can be scaled by constraining a certain percent of targets to be in the same column or row, and assigning the remaining targets randomly.

The third type of regularity, called *inherent regularity*, arises from the fact that, for each node, the same pattern needs to be repeated: turning one link on and all other links off. This type of regularity can be reduced by decreasing the number of nodes in the network, and hence the number of times that pattern needs to be repeated.

While the Bit Mirroring problem is easy to conceptualize, it is challenging for evolutionary algorithms. It requires most links to be turned off, and only a few specific links to be turned on. Moreover, links between input nodes and non-target nodes, which are likely to exist in initial random configurations and to be created by mutations, can complicate fitness landscapes. Imagine, for example, that a mutation switches the weight on a link between an input node and its target output from zero to a positive number. The organism is now closer to the ideal wiring, but it may not receive a fitness boost if other incorrect links to that output node result in the wrong net output. The Bit Mirroring problem is useful, therefore, because it is challenging, yet its three main regularities are known, and can be independently adjusted.

C. The Quadruped Controller Problem

The Quadruped Controller problem is to evolve fast gaits for simulated four-legged robots. This problem is challenging because creating dynamic gaits for legged robots is a complicated non-linear problem that is infeasible to compute mathematically; for example, effective gaits are difficult and time-consuming for human engineers to program [35], [53]. Given how sensitive gait controllers are to slight changes in the configuration of a robot, a new gait must be created each time a robot is changed, which can lead to substantial delays in the prototyping stage of robotic development [24]. It would therefore be beneficial to automate the process of gait creation.

The problem of legged locomotion also contains many regularities; each leg is a repeated module and various gaits have different regularities, such as left-right or front-back symmetry. The regularity of this problem can also be scaled by changing the number of legs that are slightly different, as can happen due to inconsistencies in manufacturing processes. The Quadruped Controller problem is thus a challenging real-world problem that has regularities that can be varied. It will help validate whether conclusions drawn from Target Weights and Bit Mirroring generalize to more challenging real-world problems.

Before describing the specific implementation of the Quadruped Controller problem in this paper, we will review previous work in this area. Many researchers have successfully evolved controllers for legged robots, typically by evolving neural network controllers [14], [19], [22], [24], [29], [39], [50], [51]. Evolved gaits are often better than those produced by human designers; one was even included on the commercial release of Sony's AIBO robotic dog [24], [51]. However, many researchers have found that evolutionary algorithms cannot handle the entire problem because the number of parameters that need to be simultaneously tuned to achieve success is large [1], [13], [24], [25], [33], [50], [51]. Many of these scientists report that, while it is possible to evolve a controller to manage the inputs and outputs for a single leg, once evolution is challenged with the inputs and outputs of many legs, it fails to make progress.

One solution that has worked repeatedly is to help the evolutionary algorithm 'see' that there are regularities and symmetries to the problem. This approach involves manually decomposing the problem by, for example, evolving a controller for one leg and then copying that controller to every other leg, with some variation in phase. Unfortunately, this tactic imposes a specific type of regularity on the network instead of allowing evolution to produce different regularities. It would be better to completely automate the process and thereby remove the need for human engineers to spend time decomposing the problem. Furthermore, such manual decomposition potentially introduces constraints and biases that could preclude the attainment of better solutions [31]. Finally, if we can employ algorithms that can automatically discover and exploit the regularities in a problem, such algorithms may be able to do the same for complex problems with regularities humans are not aware of. Indirect encodings should be well-suited to this task of automatically discovering regularities, so it is worthwhile to study their capabilities on this front.

While it has not been the norm, indirect encodings have occasionally been used to evolve the gaits of legged creatures. In at least two cases, an indirect encoding evolved both the gaits and the morphologies of creatures [23], [39]. In both cases, the morphologies and behaviors were regular. While the regularity of these ANNs was not systematically studied, one of these papers contained an anecdotal report of a regular ANN [23]. However, the regularity of the problem was not scaled in either of these works. In another study, an indirect encoding and a direct encoding were compared for their ability to evolve a gait for a legged creature in an attempt to determine whether the indirect encoding can exploit the regularity of the domain without the problem being simplified or manually decomposed [19]. However, this project used a simple model of a six-legged insect that had only two degrees of freedom per leg. Nevertheless, the work showed that the indirect encoding could automatically discover the regularity of the problem and decompose it by encoding a neural submodule once and expressing it repeatedly. The indirect encoding also outperformed a direct encoding by solving the problem faster. Unfortunately, computational limits at the time meant that such results were anecdotal and not statistically significant because so few trials could be performed.

The Quadruped Controller problem is a challenging engineering problem with scalable regularity. Investigating the performance of HyperNEAT and direct encoding controls will therefore illuminate how an indirect encoding handles problem regularities differently than direct encodings. We next describe the implementation details of the specific Quadruped Controller problem in this paper.

The robots (Fig. 6) are evaluated in the Open Dynamics Engine (ODE) physics simulator [www.ode.org]. The rectangular torso of the organism is (in arbitrary ODE units) 0.15 wide, 0.3 long, and 0.05 tall. For a point of reference, the right side of the robot from the viewer's perspective in Fig. 6 is designated as 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

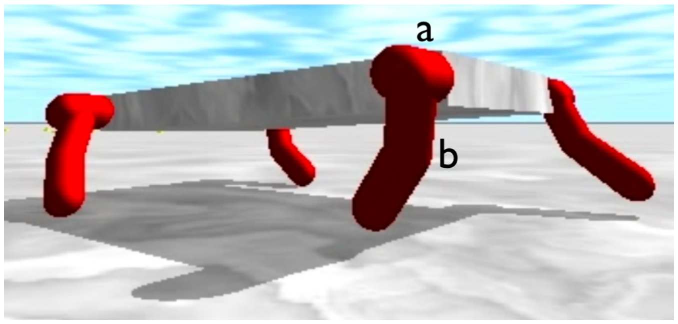

Fig. 6. The Simulated Robot in the Quadruped Controller problem. (a) Two joints (HipFB and HipIO, see text) approximate a universal hip joint that can move in any direction. (b) The knee joint can swing forward and backward.

backward (anterior-posterior) and is constrained to 180 degrees such that at maximum extension it is parallel with the torso. The second hip joint (HipIO) allows the 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 quadruped is simulated for 6 seconds (6000 time steps). Trials are cut short if any part of the robot save its lower leg touches the ground or if the number of direction changes in joints exceeds 960. The latter condition is an attempt to roughly reflect that servo motors cannot be vibrated incessantly without breaking. The fitness of a controller is

$$fitness = 2^{d^2}, \tag{3}$$

where d is the maximum distance traveled during the allotted time. An exponential fitness function is used so that even small increases in the distance traveled result in a sizable selective advantage.

For HyperNEAT and FT-NEAT, the ANN configuration on this problem features three two-dimensional, 5×4 Cartesian grids forming input, hidden and output layers (Fig. 7). The NEAT control has the same number of inputs and outputs, but the number of hidden nodes can evolve. There are no recurrent connections. All possible connections between adjacent layers exist (although weights can be zero, functionally eliminating the link). There are thus 800 links in the ANN of each individual for HyperNEAT and FT-NEAT. The number of links in NEAT can evolve. Link weights are in the range of $[-3, 3]$.

The inputs to the ANN are the current angles (from $-\pi$ to π) of each of the 12 joints of the robot, 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 facilitates periodic behavior). The sine wave function is

$$sin(t/120)\pi, \tag{4}$$

where t is the number of milliseconds that have passed since the start of the experiment. Multiplying by π produces numbers between $-\pi$ and π, which is the range of the unconstrained joints. The constant 120 was chosen because it was experimentally found to produce fast yet natural gaits. While changing this constant can affect the types of gaits produced,

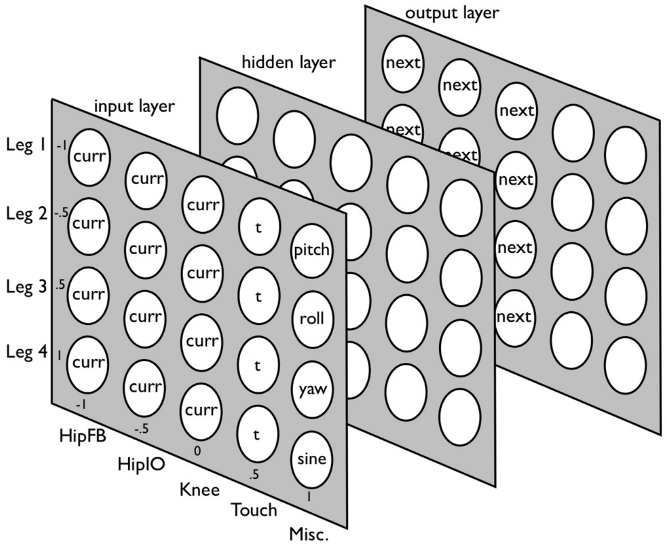

Fig. 7. ANN Configuration for HyperNEAT and FT-NEAT Treatments. The first four columns of each row of the input layer receive information about a single leg (the current angle of each of its three joints, and a 1 or 0 depending on whether the lower leg is touching the ground). The final column provides the pitch, roll, and yaw of the torso as well as a sine wave. Evolution determines how to use 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 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.

doing so never altered any of the qualitative conclusions of this paper. Preliminary tests determined that the touch, pitch, roll, yaw, and sine inputs all improved the ability to evolve fit gaits.

The outputs of the ANNs are the desired joint angles for each joint, which are fed into a PID controller that simulates a servo. The controller subtracts the current joint angle from the desired joint angle. This difference is then multiplied by a constant force (2.0), and a force of that magnitude is applied to the joint such that the joint moves toward the desired angle. Such PID-based control systems have been shown to be effective in robot control [24], [29], [54].

Regularity in the Quadruped Controller problem can be scaled by changing the number of faulty joints. A faulty joint is one in which, if an angle A is requested, the actual desired angle sent to the PID controller is $A + E$, where E is an error value in degrees within the range $[-2.5, 2.5]$. The value of E is chosen from a uniform random distribution in this range for each faulty joint at the beginning of a run, and is constant throughout the run. Such errors are analogous to the inconsistencies of robotic joints produced by manufacturing processes. The more faulty joints, the less regularity there is in the problem because fewer legs behave identically. The default version of this problem [5] is the most regular version with zero faulty joints, which is the version referred to throughout the rest of the paper unless the regularity is specified. Each algorithm was run 50 times for experiments with 0, 1, 4, 8, and 12 faulty joints. Runs lasted 1000 generations and had a population size of 150.

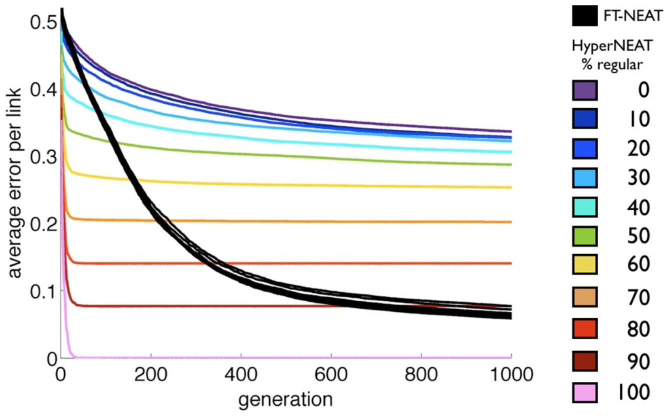

Fig. 8. Mean Performance of HyperNEAT and FT-NEAT on a Range of Problem Regularities for the Target Weights Problem. HyperNEAT lines are colored for each regularity level, and in early generations are perfectly ordered according to the regularity of the problem (i.e., regular treatments have less error). The performance of FT-NEAT (black lines) is unaffected by the regularity of the problem, which is why the lines are overlaid and mostly indistinguishable.

IV. Results

The following sections describe experiments and analyses that investigate how the HyperNEAT generative encoding compares to direct encodings with respect to exploiting problem regularities and producing phenotypic regularities.

A. HyperNEAT Outcompetes Direct Encodings as Problem Regularity Increases

1) Target Weights: The results from the Target Weights experiments (Fig. 8) reveal that HyperNEAT performs better as the regularity of the problem increases, especially in early generations, where mean performance perfectly correlates with problem regularity. Interestingly, after 1000 generations of evolution, the performance of HyperNEAT is statistically indistinguishable below a certain regularity threshold ($p > 0.05$[1] comparing the final performance of the $S = 0\%$ treatment to treatments with $S \leq 30\%$). Above that regularity threshold, however, HyperNEAT performed significantly better at each increased level of regularity ($p < 0.01$ comparing treatment with values of $S > 30\%$ to the treatment with a value of S 10% higher). FT-NEAT, on the other hand, is blind to the regularity of the problem: the results from different treatments are visually and statistically indistinguishable ($p > 0.05$). Early on HyperNEAT outperformed FT-NEAT on regular versions of the problem ($p < 0.01$ comparing treatments of $S \geq 60\%$ at generation 100), but at 1000 generations FT-NEAT outperforms HyperNEAT on all but the two most regular versions of the problem ($p < 0.001$). HyperNEAT outperforms FT-NEAT on the most regular version of the problem at generation 1000 ($p < 0.001$), and the algorithms are statistically indistinguishable on the $S = 90\%$ treatment ($p > 0.05$).

Overall, this experiment provides evidence for several interesting observations. While this evidence comes from only the Target Weights problem, which is a simple diagnostic problem, we highlight them here because they will also be supported by data from the Bit Mirroring and Quadruped Controller problems. They are as follows:

- FT-NEAT outcompetes HyperNEAT when problem regularity is low.
- As problem regularity increases, HyperNEAT's performance rises to, and then surpasses, that of FT-NEAT, demonstrating that HyperNEAT can exploit problem regularity.
- FT-NEAT is blind to problem regularity.
- HyperNEAT exploits problem regularity only above a certain regularity threshold.

A final result of interest from this experiment is the lack of progress HyperNEAT makes after the early generations on the problems that are mostly regular, but have some irregularity (e.g., $S = 90\%$). HyperNEAT is easily able to produce weights similar to the repeated weight Q, and thus exploits the regularity of the problem, but over hundreds of generations HyperNEAT did not discover how to make exceptions to the pattern to produce the irregular link values. This evidence suggests HyperNEAT is biased toward producing regular solutions and has difficulty producing certain irregular patterns, a subject we will revisit later in the paper.

2) Bit Mirroring: The first two Bit Mirroring experiments have 7×7 grids of input and output nodes. In the first experiment, both within-column and within-row regularity start at 100%, and within-column regularity decreases per treatment by constraining fewer targets to be in the same column as their respective inputs (Type I in Fig. 5b). The most irregular treatment in this experiment features zero within-column regularity, but has 100% within-row regularity. The second experiment picks up where the first left off by lowering within-row regularity per treatment (Type II in Fig. 5b). The least regular treatment in experiment two has no within-column or within-row regularity. For each treatment, 10 runs of evolution are performed.

The results from the Bit Mirroring experiment support many of the conclusions from the Target Weights experiment. Initially, FT-NEAT is blind to both within-column and within-row regularity (Fig. 9, right two columns, $p > 0.05$ comparing all within-column and within-row treatments to the treatments with no column or row constraints, respectively). The performance of HyperNEAT, however, increases with the regularity of the problem (Fig. 9, left two columns). HyperNEAT perfectly solves the problem in all but two treatments on the most regular version of the problem, where targets are in the same column and row. Once again, the performance of HyperNEAT within a type of regularity does not increase until that type of regularity is above a threshold. For both within-column and within-row regularity, HyperNEAT's performance advantage is statistically significant only once that type of regularity is above 50% (only treatments with more than 50% of targets column-constrained or row-constrained statistically outperform treatments with 0% of targets column-constrained or row-constrained, respectively: $p < 0.05$).

It is also interesting that the magnitude of the *range* of

[1]This p value and all others in this paper were generated with Matlab's non-parametric Mann-Whitney U-test, unless otherwise specified.

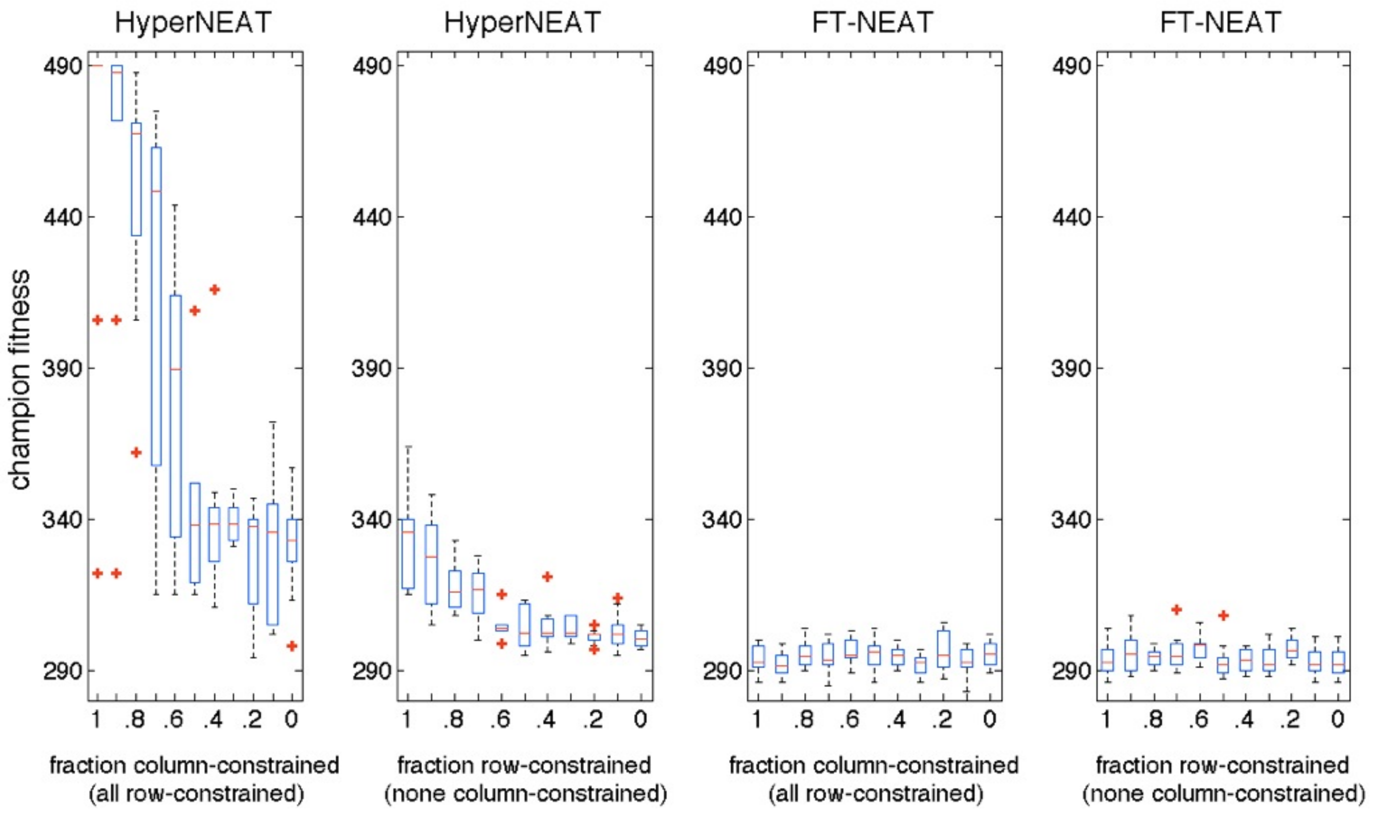

Fig. 9. HyperNEAT and FT-NEAT on Versions of the Bit Mirroring Problem with Different Levels of Regularity. For each treatment, from left to right, within-column regularity is first decreased (left panel) and then within-row regularity is further decreased (right panel). The data plotted are collected from populations at the end of evolutionary runs, but plots over time (not shown) reveal the same qualitative story: HyperNEAT's performance is consistently higher on more regular problems and the performance of FT-NEAT is unaffected by the regularity of the problem. For HyperNEAT, the performance gaps between more and less regular treatments are evident early and increase across evolutionary time.

fitness values is correlated with the regularity of the problem for HyperNEAT. This phenomenon might occur because, when regularity is present, the indirect representation either discovers and exploits it, which would result in high fitness values, or it fails to fully discover the regularity, at which point its fitness more closely resembles lower-performing, less-regular treatments.

HyperNEAT outperforms FT-NEAT in all versions of this problem ($p < 0.05$). This result is likely due to the inherent regularity of the problem, which arises because the same pattern (turning one link on and the rest off) must be repeated for each of the 49 input nodes. Inherent regularity should decrease with the number of nodes in the network, a hypothesis we test in experiment three, where experiments are performed on grid sizes from 8×8 down to 3×3 (Fig. 10). For all grid sizes, both within-column and within-row regularity levels are 0%, leaving only the inherent regularity of the problem. Due to the high variance between runs, 40 runs are conducted per treatment.

As the grid size is lowered, the relative performance of HyperNEAT degrades to and then falls below that of FT-NEAT. The general trend is significant ($p < 0.05$ comparing the ratios on the 3×3 problem versus those with 6×6 and larger grids). It is not clear why HyperNEAT performed relatively better on the 3×3 grid than the 4×4 grid. This experiment reinforces the conclusion that once problems become irregular enough, FT-NEAT can outperform HyperNEAT. It also provides a further demonstration that HyperNEAT can exploit increasing problem regularity to gain a relative edge over FT-NEAT.

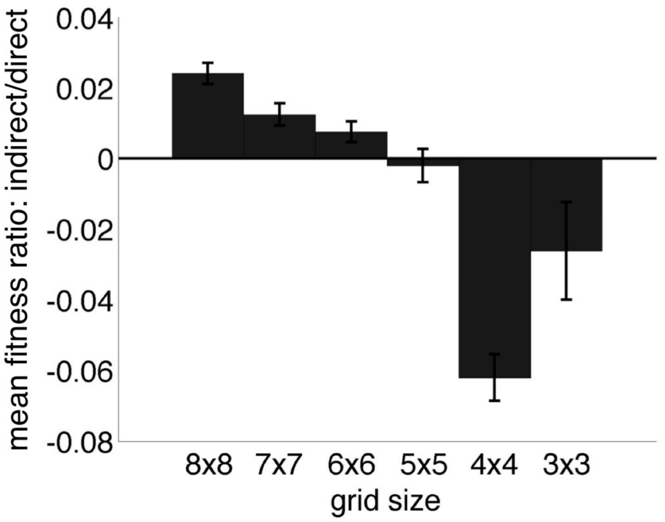

Fig. 10. HyperNEAT Vs. FT-NEAT as the Inherent Regularity of the Bit Mirroring Problem is Decreased. Reducing the grid size reduces the amount of inherent regularity in the problem. Error bars show one standard error of the mean. Ratios are used instead of absolute differences because the allowable fitness ranges change with grid size.

3) Quadruped Controller: The data from the Quadruped Controller problem also generally support the conclusions from Target Weights and Bit Mirroring. HyperNEAT outperforms both FT-NEAT and NEAT on the two most regular versions of this problem, where there are 0 or 1 faulty joints (Fig. 11, $p < 0.001$). That HyperNEAT outperforms NEAT is noteworthy, given that NEAT is one of the most successful direct encoding neuroevolution algorithms.

As with Target Weights and Bit Mirroring, HyperNEAT's performance increases with the regularity of the problem, but only above a certain threshold: the 0 faulty joint treatment significantly outperforms the 1 faulty joint treatment ($p < 0.001$) which, in turn, outperforms the 4 faulty joint treatment ($p < 0.001$) which, in turn, outperforms the 8 faulty joint treatment ($p < 0.001$). However, the 8 and the 12 faulty joint treatments are statistically indistinguishable ($p > 0.05$). In contrast to Target Weights and Bit Mirroring, FT-NEAT is not blind to the regularity of this problem, although it is less sensitive to the regularity than HyperNEAT. The treatment with 0 faulty joints is statistically indistinguishable from the 1 faulty joint treatment ($p > 0.05$), but performance on both of these treatments is higher than on the 4 faulty joint treatment ($p < 0.001$) which is, in turn, higher than the 8 faulty joint treatment. As is the case for HyperNEAT, performances for FT-NEAT on the treatments with 8 and 12 faulty joints are statistically indistinguishable ($p > 0.05$). The statistical comparisons for NEAT are the same as those for FT-NEAT.

One reason that regularity may affect the direct encodings on this problem is because weights tend to be near the maximum or minimum allowable value, which is partly because mutations that create links outside of this range are set to the maximum or minimum value. This method makes extreme values more likely, which can facilitate coordination in the joints because different joints are controlled by links with similar weights. If normal joints are controlled by links with the maximum or minimum link value, to have faulty joints behave the same as normal joints, link values would either have to be outside the allowable range, or inside the range at non-extreme (and thus harder to set) values. Faulty joints thus increase the difficulty of the problem for both indirect and direct encodings, and can help explain why the problem regularity appears to benefit the direct encodings. Exploring ways to reduce this bias is an interesting avenue for future work.

As with Bit Mirroring and Target Weights, FT-NEAT is able to outperform HyperNEAT on the Quadruped Controller problem once the regularity of the problem is sufficiently low. FT-NEAT and NEAT outperformed HyperNEAT on both the 8 and 12 faulty joint treatments. On the treatment with 8 faulty joints, the difference is significant for FT-NEAT ($p < 0.05$), but not for NEAT. On the treatment with 12 faulty joints, the difference for FT-NEAT is almost significant ($p = 0.066$), but the difference for NEAT is highly significant ($p < 0.01$).

The difference in fitness in the first generation of randomly-generated organisms is interesting. HyperNEAT begins with an advantage over FT-NEAT because even randomly-generated CPPNs are sometimes able to produce the coordination of legs that facilitates movement. Some of these randomly-generated organisms in HyperNEAT display impressive coordination and appear to be on the road toward rudimentary locomotion. Randomly-generated FT-NEAT and NEAT organisms do not provide this impression.

Overall, the results from the Target Weights, Bit Mirroring, and Quadruped Controller problems show that the direct encodings outperform HyperNEAT when problem regularity is low. They also show that as problem regularity increases, HyperNEAT can exploit that regularity whereas the direct encodings mostly do not. This ability to exploit problem regularity means that HyperNEAT increasingly outperforms direct encoding controls as problem regularity increases. We now investigate further how HyperNEAT is able to exploit regularity.

B. HyperNEAT Produces More Regular Behaviors

We focus our analysis of regularity in ANNs and behaviors on the Quadruped Controller problem because Target Weights and Bit Mirroring are diagnostic problems wherein regularity is explicitly built into the problem (i.e., any phenotypic regularity in fit solutions is unsurprising). Moreover, there is no meaningful behavior associated with the two diagnostic problems. The Quadruped Controller problem, on the other hand, does have interesting behaviors with different levels of regularity. Moreover, the problem does not explicitly require or reward regularity, which means that any regularities that develop do so because of the encoding and because such regularities happen to produce fast gaits.

The first method we employ to analyze the behaviors produced by the different algorithms is based on videos of the highest performing gaits from all 50 runs in the treatment with 0 faulty joints (available at http://devolab.msu.edu/SupportDocs/Regularity). The HyperNEAT gaits are all regular. They feature two separate types of regularity: coordination between legs, and repetition of the same movement pattern across time. Generally, the gaits are one of two types. The first type has four-way symmetry,

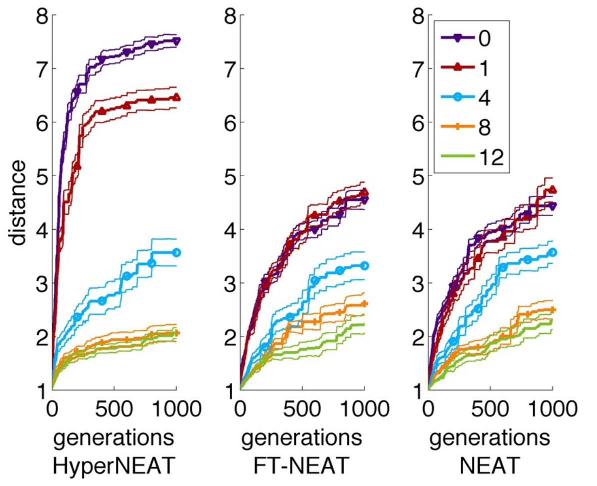

Fig. 11. Performance of HyperNEAT, FT-NEAT, and NEAT on the Quadruped Controller Problem with 0, 1, 4, 8, and 12 faulty joints.

wherein each leg moves in unison and the creature bounds forward repeatedly (Fig. 12, top row). This gait implies that HyperNEAT is reusing neural information in a regular way to control all of the robot's legs. The second gait resembles a horse gallop and features the back three legs moving in unison, with the fourth leg moving in opposite phase. This *3-1 gait* demonstrates that HyperNEAT can reuse neural information with some variation, because the same behavioral pattern exists in each leg, but is inverted in one leg. The ability to produce repetition with variation is a desirable feature in genetic encodings [46].

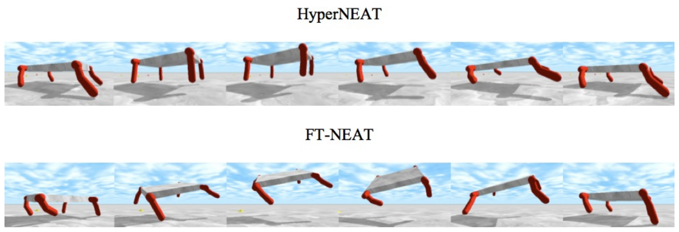

Fig. 12. A Time Series of Images from Typical Gaits Produced by HyperNEAT and FT-NEAT. HyperNEAT robots typically coordinate all of their legs, whether all legs are in phase (as with this robot) or with one leg in anti-phase. A short sequence involving a bound or gallop is repeated over and over in a stable, natural gait. FT-NEAT robots display far less coordination among legs, are less stable, and do not typically repeat the same basic motion. NEAT gaits are qualitatively similar to FT-NEAT gaits.

Overall, the HyperNEAT gaits resemble those of running natural organisms because they are coordinated and graceful. These observations are noteworthy because they indicate that HyperNEAT is automatically exploiting the regularities of a challenging, real-world problem. This accomplishment is significant given that researchers have previously needed to manually decompose legged locomotion tasks for evolutionary algorithms to perform well [1], [13], [24], [25], [33], [50], [51].

The gaits of FT-NEAT and NEAT, on the other hand, are mostly uncoordinated and erratic, with legs often appearing to operate independently of one another (Fig. 12, bottom row). A few of the best performing gaits do exhibit coordination between the legs, and the repetition of a basic movement pattern, but most of the gaits are irregular. Even the regular gaits are not as natural and impressive as the HyperNEAT gaits, which is reflected in their lower objective fitness values. For most gaits, some legs flail about, others trip the organism, and some work against each other by pushing in opposite directions. The robots frequently cartwheel and trip in unstable positions until they finally fall over. There is much less repetition of a basic movement pattern across time. Coordination between legs is often rare and temporary.

The few examples of regular gaits produced by FT-NEAT and NEAT show that it is sometimes possible for direct encodings to produce regularities. Overall, however, HyperNEAT is much more consistent at producing regular gaits. All of the HyperNEAT gaits are regular, whereas only a few FT-NEAT and NEAT gaits are. It is important for algorithms to be consistent, especially when computational costs are high, so that high-quality results can be obtained without performing many runs. A test of the reliability of each encoding is to watch the median and least fit gaits of the 50 champions for each encoding: for HyperNEAT these gaits are coordinated and effective, whereas for FT-NEAT and NEAT they are discombobulated.

In general, the gaits reveal a greater gap in performance between HyperNEAT and the direct encodings than is suggested by the fitness scores, especially for all but the best runs for each algorithm. Most of the direct encoding gaits do not resemble stable solutions to quadruped locomotion, whereas HyperNEAT produces a natural gait in all trials with a small variety of different solutions.

A second method for investigating how HyperNEAT is able to outperform the direct encodings is to look at the angles of the leg joints during locomotion. This technique is a different way of estimating the coordination, or lack thereof, of the different legs for each encoding. Plots of each leg's HipFB joint from the best, median, and worst runs for each algorithm corroborate the descriptive evidence (Fig. 13). The legs in all HyperNEAT organisms exhibit a high degree of both of the two main regularities: at any point in time most legs are in similar positions (except the exception leg in the 3-1 gait, which is opposite), and a basic movement pattern is repeated across time. The direct encoding gaits are less regular in both ways, except for the highest performing gaits. The median and worst gaits are representative of most of the direct encoding gaits: there is little coordination between legs or across time (Fig. 13). While only the HipFB joint is shown, plots of the other two joints are consistent with these results.

C. HyperNEAT Behaviors are More General

One of the benefits of regularity is generalization. On the Quadruped Controller problem, for example, repeating the same basic pattern of motion is a type of regularity that is likely to generalize, because its success in one cycle makes it probable that it will be successful in the next cycle. A non-repeating sequence of moves, however, may be less likely to generalize because its past is less likely to predict its future. Generality, then, can be a test of regularity. It is also a desirable property in its own right.

During the evolution experiment, the robots are evaluated for six simulated seconds. One test of generality is to remove the time limit of six seconds and measure how long the evolved robots are able to move before they fall. Fig. 14 reports that HyperNEAT champion gaits are significantly more general than FT-NEAT and NEAT champion gaits ($p < 0.001$). HyperNEAT gaits are the only ones on average that keep moving beyond the number of seconds simulated during evolution.

D. HyperNEAT Regularities can be Influenced, Allowing Domain Knowledge to be Injected

HyperNEAT genomes create regularities in geometric space that affect phenotypic elements based on the geometric coordinates of those elements. Changing the geometric arrangement of these elements may make it easier for HyperNEAT to produce one type of behavior versus another [8]. For example, it might be easier to group two elements together if those elements are close to each other, whereas this grouping may

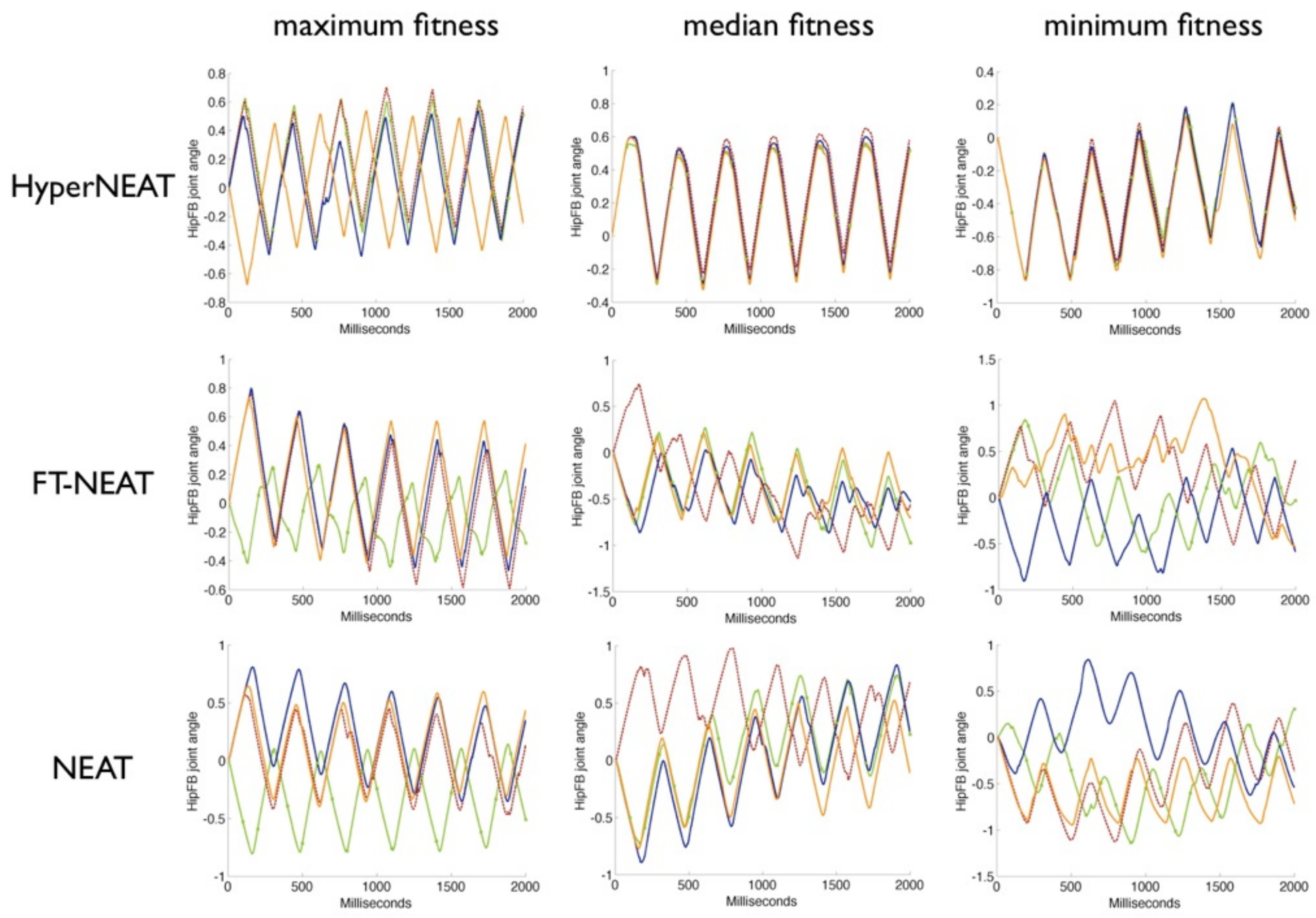

Fig. 13. HipFB Joint Angles Observed in Robots Evolved with HyperNEAT, FT-NEAT, and NEAT. The possible range for this joint is -0.5π to 0.5π. The Y axis shows radians from the initial down (0) position. For clarity, only the first 2 seconds are depicted. For HyperNEAT, the best gait is an example of the 3-1 gait, where three legs are in phase and one leg is in opposite phase, which resembles the four-beat gallop gait. The other two HyperNEAT gaits are four-way symmetric, with all legs coordinated in a bounding motion (Fig. 12). The best direct encoding gaits are mostly regular. However, the median and worst gaits, which are representative of most direct encoding gaits, are irregular: while some legs are synchronized, other legs prevent the coordinated repetition of a pattern.

be more difficult if the elements are far away from each other, especially if elements that should not be included in the group lie in-between. If this hypothesis is correct, arranging phenotypic elements such as sensors or outputs in different geometric configurations may be a way for the experimenter to influence the types of regularities HyperNEAT produces.

The ordering of the legs in HyperNEAT for the Quadruped Controller problem offers a way to test this hypothesis. In the default configuration (Fig. 7) the legs are ordered, from lowest to highest Y coordinate value, FL-BL-BR-FR, where F=Front, B=Back, L=Left and R=Right. This ordering may make it easier to group the left legs into one group and the right legs into another, since they are closer to each other. To test this hypothesis, we performed experiments with the following alternate orderings: FL-FR-BL-BR and FL-FR-BR-BL, which may encourage front-back symmetry, and FL-BR-FR-BL, which may encourage diagonal symmetry. For each of these four orderings, we performed 50 runs, each with a population size of 150 that lasted 1000 generations. Table I reports the classifications of the highest performing gait at the end of each run.

	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

TABLE I
THE RESULTANT GAIT TYPES FOR DIFFERENT LEG ORDERINGS. GAITS ARE 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), AND ONE LEG OUT OF PHASE (THREE LEGS MOVED IN SYNCHRONY AND ONE IS OUT OF PHASE, WHICH RESEMBLES A GALLOP). IF TWO LEGS ARE MOTIONLESS, THEY ARE CONSIDERED IN SYNCHRONY. TWO GAITS DO NOT FIT INTO THESE CATEGORIES AND ARE NOT TABULATED. FL=FRONT LEFT, BL=BACK LEFT, BR=BACK RIGHT AND FR=FRONT RIGHT.

The most common gait in all runs exhibits four-way symmetry, which is not expected to be biased by leg ordering. The

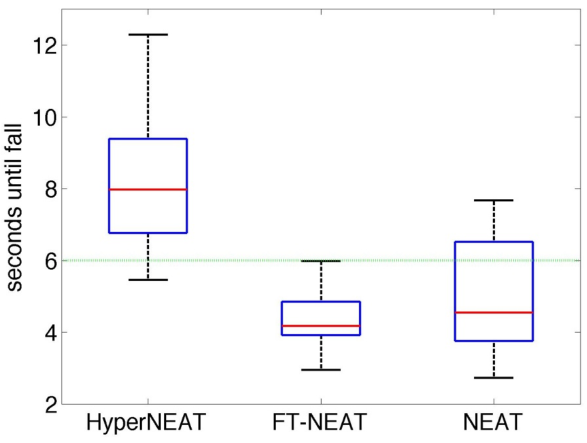

Fig. 14. Gait Generalization. HyperNEAT gaits generalize better than FT-NEAT and NEAT gaits, which means that they run for longer before falling over. The allotted time during evolution experiments is 6 seconds (dashed horizontal line). Only the HyperNEAT gaits exceed that amount of time in the generalization tests. For clarity, outliers are not shown.

other gaits, however, do tend to reflect the geometric ordering of the legs in each treatment. For example, all four examples of left-right symmetry evolved in the *L*L*R*R treatment (where * stands for any symbol), and all seven cases of front-back symmetry evolved in the treatments that ordered the legs F*F*B*B*. It appears it is easier for HyperNEAT to bisect the Y dimension once to group neighboring legs, instead of creating the more complex pattern required to group legs with non-adjacent Y coordinate values.

It is interesting to observe which is the exception leg in the gaits that had three legs in synchrony and one leg in opposite phase. In 23 out of 25 cases, the exception leg is the one with the highest Y coordinate value, although which leg that is changes based on the geometric ordering (Table I). Different geometric representations, therefore, can probabilistically bias evolution to make different legs be the exception leg, which is an example of how a HyperNEAT user can inject a preference into the algorithm. It is not clear why exceptions are typically made for the leg with the highest Y coordinate value. This result may be due to the nature of the mathematical functions in the CPPNs.

E. HyperNEAT ANNs are More Regular, which is Visually Apparent

Beyond behavioral regularities, it is also interesting to examine the ANNs produced by HyperNEAT and FT-NEAT. NEAT ANNs are not visualized because their variable number of hidden-node layers make such visualization difficult. Visualizations reveal that all HyperNEAT ANNs are regular while all FT-NEAT ANNs are irregular. Fig. 15 shows example ANNs produced by HyperNEAT and FT-NEAT (all 50 visualizations for HyperNEAT and FT-NEAT ANNs can be viewed at http://devolab.msu.edu/SupportDocs/Regularity). The FT-NEAT ANN has no obvious regularity, and is hard to visually distinguish from an ANN with randomly-generated weights. Such ANN irregularity occurs even when the behavior produced by the ANN is somewhat regular (e.g., despite producing the relatively regular gait seen in the left-most column of Fig. 13, the FT-NEAT ANN in Fig. 15 is irregular).

Because nodes do not have defined geometric coordinates in FT-NEAT, it would be equally appropriate to visualize its ANNs with any rearrangement of nodes within a layer. In this paper, all FT-NEAT nodes are plotted in the same arbitrary order. While different orderings may affect the visual appearance of regularity, the experimental evidence suggests that it is unlikely that reordering the nodes will reveal substantial regularity in the FT-NEAT ANNs.

The HyperNEAT ANN in Fig. 15, on the other hand, displays multiple different regularities. Initially, looking from the front, the links emanating from each input node repeat a pattern with inhibitory (dark red) links on top and excitatory (light green) links below. Additionally, a clear distinction is made between the different layers of connections, at least on the top of the ANN, where the input-to-hidden layer is inhibitory and the hidden-to-output layer is excitatory. An additional regularity is observable (viewing from the back) in links projecting from the outputs, where the nodes toward the bottom-left corner have a pattern of a few inhibitory links surrounded by excitatory links. Interestingly, this pattern is repeated in nearby nodes, but the strength of the pattern decreases roughly with the distance from the bottom-left corner, such that eventually there are few or no remaining inhibitory links. The decrease is faster in the Y (height) dimension, and slower in the X (width) dimension. This pattern shows that HyperNEAT can produce complex geometric regularities.

A diverse set of regularities are observable in the 50 HyperNEAT champion ANNs. Some of this diversity is shown in Fig. 16 and Fig. 17. Left-right, top-bottom, and diagonal symmetries are produced. Additionally, exceptions (different patterns) are made for single columns, rows, and individual nodes. Some networks appear completely regular, with every viewable link having a similar value, and others feature complicated, yet still regular, patterns. Some networks have predominantly large weights (e.g., the top-left ANN in Fig. 16), others have mainly small weights (e.g., the top-center ANN in Fig. 16), and some have a broad range of weights. Importantly, many of these regularities include some variation. For example, the top-left ANN in Fig. 16 has a motif that is repeated for each node, but that motif varies in regular ways (e.g., the number and strength of excitatory links emanating from nodes in the second column increases from bottom to top). Other variation is less regular, such as the exception made for a single node (Fig. 16, bottom left), but even in this case the node itself contains a regular pattern.

This diversity of regularities demonstrates that HyperNEAT can explore many different types of regularities to solve a problem. It is also interesting that, while HyperNEAT is able to find many different regular solutions that perform and behave similarly, FT-NEAT is unable to consistently discover *any* of these regular patterns. In other words, there are many solutions in the search space, but the regularity of those solutions means that an indirect encoding frequently encounters them while the direct encoding rarely does.

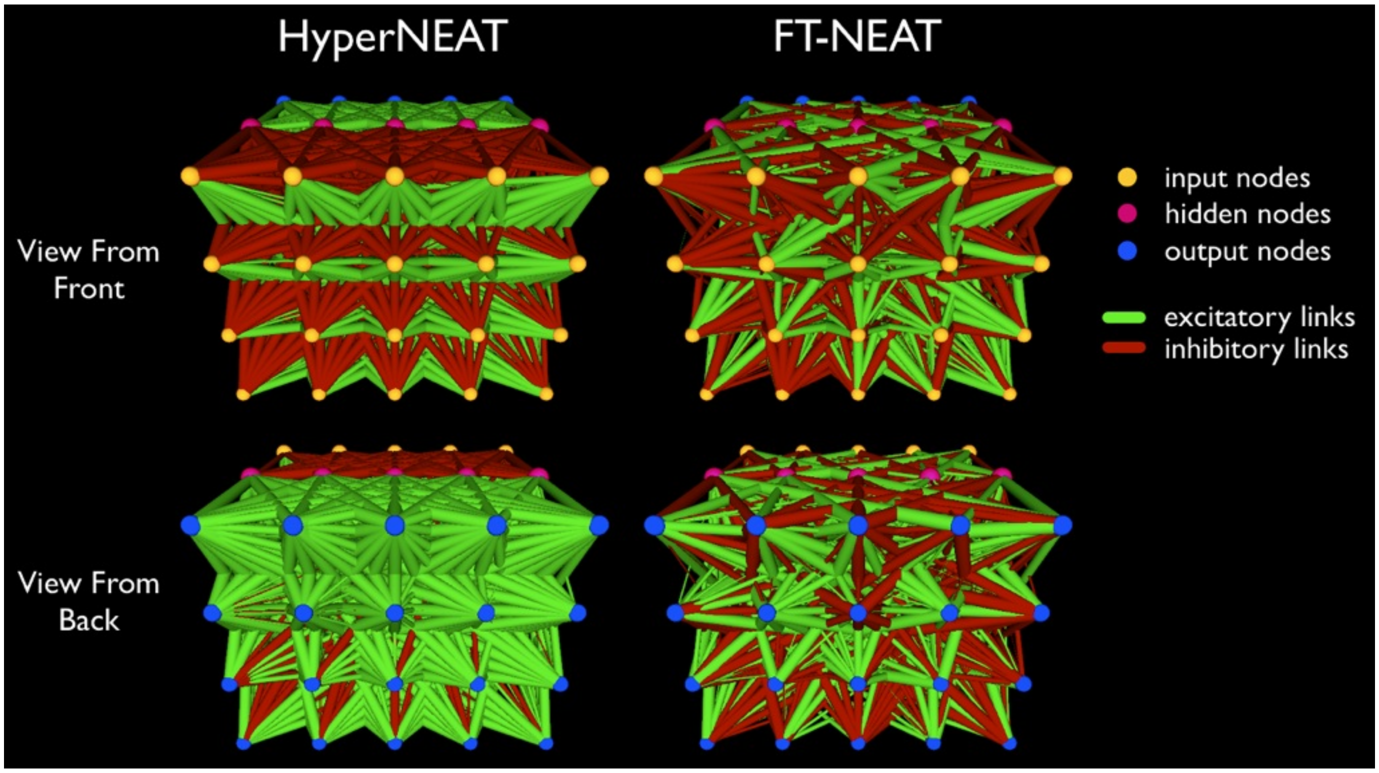

Fig. 15. Example ANNs Produced by HyperNEAT and FT-NEAT. Views from the front (looking at the inputs) are shown in the top row, and from the back (looking at the outputs) are shown in the bottom row. The thickness of the link represents the magnitude of its weight.

As mentioned previously, HyperNEAT predominantly produces two different gaits on the Quadruped Controller problem: one with all legs in synchrony and the other with three legs in synchrony and the exception leg in opposite phase. It is sometimes possible to infer the behavior of a robot just by viewing a visualization of the regularities in its neural wiring. For example, ANNs that produce four-way symmetric gaits display similar wiring patterns for each row of outputs (Fig. 17, top). Recall that each row of outputs controls a separate leg (Fig. 7). For the 3-1 gaits in this experiment, the exception leg is always the front right leg, which is controlled by outputs in the top row. ANNs that produce 3-1 gaits frequently have a different pattern of weights for the top row of outputs than for the other outputs (Fig. 17, bottom). While it is not always possible to classify gaits by viewing ANNs alone, an informal experiment by one of the authors made a correct prediction for most runs.

It is interesting to note that in most output rows, patterns generalize to output nodes that do not control any aspect of the robot, and are therefore irrelevant to selection. Recall that in the output layer, only nodes in the first three columns (counting from the right when viewing from the back) control joints in the robot. The outputs in the last two columns are ignored (Fig. 7). This design decision enables us to investigate whether the patterns that HyperNEAT evolves generalize to geometric areas that are not under selective pressure. In a direct encoding, one would expect that weights connected to nodes that are not under selective pressure would be random. Accordingly, these unused columns do appear random in the FT-NEAT ANN visualizations, but so do all of the columns (Fig. 15). The patterns for unused HyperNEAT output nodes, on the other hand, generally exhibit extensions of the patterns displayed across the entire network. This phenomenon occurs in nearly every case, although a few networks had a different pattern in unused nodes than in other nodes within the same row (e.g., the bottom-right ANN in Fig. 17).

This result suggests that if joints or legs were added to an evolved HyperNEAT ANN, HyperNEAT would likely produce similar behaviors in those new joints and legs as elsewhere in the robot. If that is right, HyperNEAT would also likely succeed more consistently than a direct encoding when evolving a pre-evolved ANN further to control additional legs or joints, provided that it is beneficial to have behavioral similarities between the original and newly added components. This prediction assumes that the new components are placed at geometric coordinates such that an appropriate regularity applies. This general idea has already been demonstrated to work in a domain where HyperNEAT evolved ANNs to control multiple agents of a team [9]. Such an ability to transfer skills from one task to a new task is an important goal of machine learning [49] that an indirect encoding like HyperNEAT can potentially perform well at [52], but where a direct encoding is likely to perform poorly.

F. HyperNEAT ANNs are Quantifiably More Regular

Because regular structures require less information to describe them, regularity can be measured by compression [28]. One test of the regularity of a structure, then, is how much it can be compressed. We applied the standard unix gzip

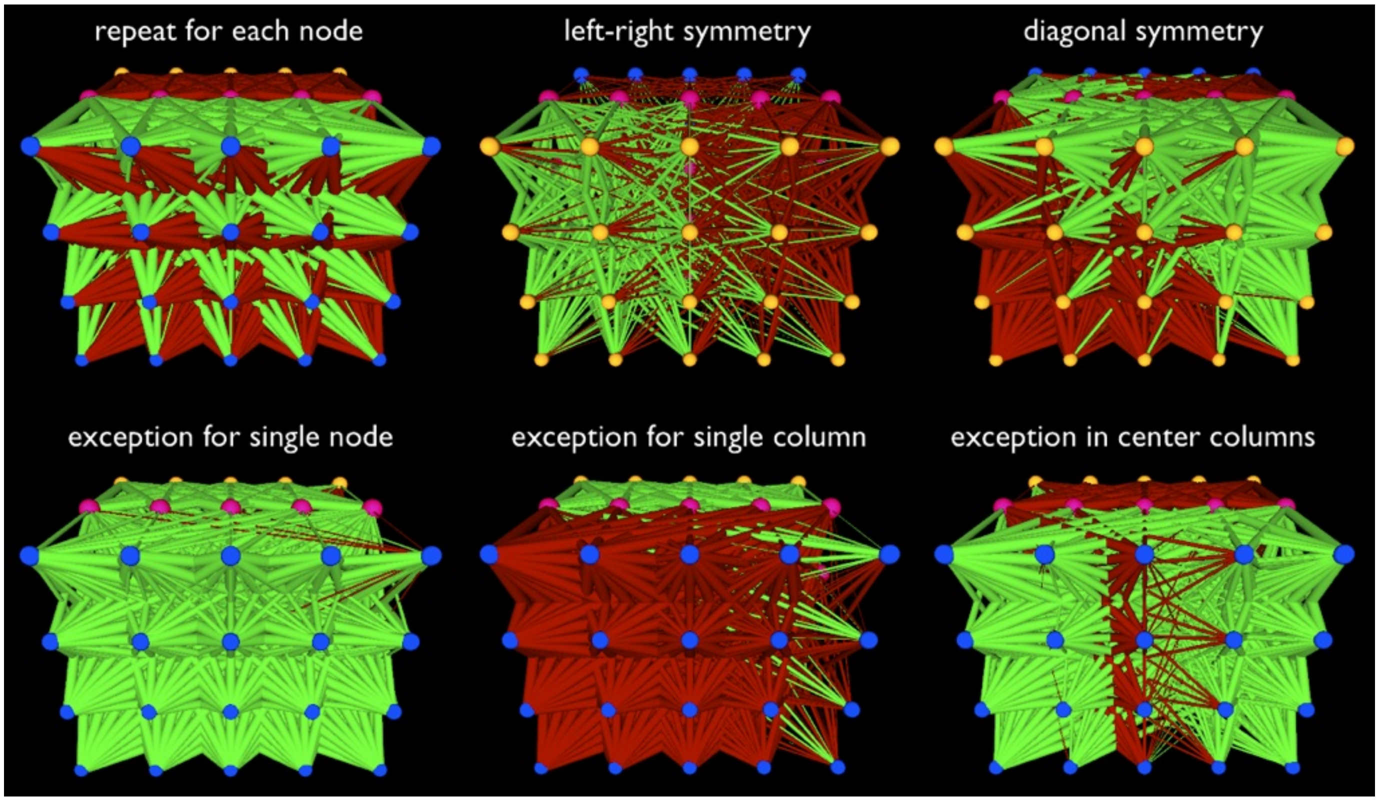
repeat for each node
left-right symmetry
diagonal symmetry
exception for single node
exception for single column
exception in center columns

Fig. 16. A Diverse Set of ANN Regularities Produced by HyperNEAT, with and without Variation. Fig. 15 explains what different colors/shades and line thicknesses represent.

compression program (version 1.3.5) to text files that contain only the weights of each ANN phenotype for HyperNEAT and FT-NEAT. NEAT ANNs are not compared because the number of links in NEAT ANNs can vary, which means that compressibility alone is not isolated. This analysis is performed on the Quadruped Controller problem because it does not have regularity explicitly built in, as opposed to Target Weights and Bit Mirroring, where fitness scores already indicate ANN regularity.

The gzip algorithm is a conservative test of regularity because it looks for repeated symbols, but does not compress all mathematical regularities (e.g., each link weight increasing by a constant amount). Nevertheless, gzip is able to compress HyperNEAT ANNs on the regular Quadruped Controller problem significantly more than FT-NEAT ANNs: $p < 0.001$, comparing the difference between each uncompressed and compressed HyperNEAT file (mean 4488 bytes ± 710 SD) and each FT-NEAT file (mean 3349 bytes ± 37 SD). This quantifiable result confirms the clear yet subjective observation from visually inspecting HyperNEAT and FT-NEAT ANNs (Fig. 15), namely, that HyperNEAT ANNs are more regular.

G. The Regularity Of HyperNEAT ANNs Correlates with the Regularity of the Problem

Another measure of the regularity of HyperNEAT ANNs is the number of nodes or links in the CPPN genome. Some of the HyperNEAT end-of-run champions on the Quadruped Controller problem, for example, have as few as 9 nodes and 26 links in their genome. Given that this genome encodes an ANN with 60 nodes and 800 links, the compression in this case is 6.6-fold and 30.8-fold, respectively. Unfortunately, this measure of regularity cannot be used for comparisons between HyperNEAT and direct encodings. It is unfair to compare genome sizes between HyperNEAT and FT-NEAT, given that the FT-NEAT genome is the same size as the final ANN. A comparison to NEAT is similarly unfair, because its genomes at least need to contain one node for every input and output node in the final ANN.

It is interesting, however, to investigate whether HyperNEAT genomes are smaller on more regular problems. As expected, the number of CPPN nodes in end-of-run champions tends to increase with the irregularity of the problem, meaning there is more compression (i.e., smaller genomes) on more regular problems. We focused this analysis on genomic *nodes* only, because the number of genomic links is correlated with the number of nodes. In Target Weights, the correlation between problem irregularity and number of CPPN nodes is positive ($r = 0.54$), although the trend is slightly insignificant ($p = 0.08$, using Matlab's corrcoef correlation coefficients test). In Bit Mirroring, the correlation is more dramatic (Fig. 18). For both the experiment that reduces column regularity, and the experiment that further reduces row regularity, the trend is positive ($r = 0.91$) and highly significant ($p < 0.001$, using Matlab's corrcoef correlation coefficients test). On the Quadruped Controller problem the trend is also positive ($r = 0.58$), although insignificant ($p > 0.05$, using Matlab's corrcoef correlation coefficients test).

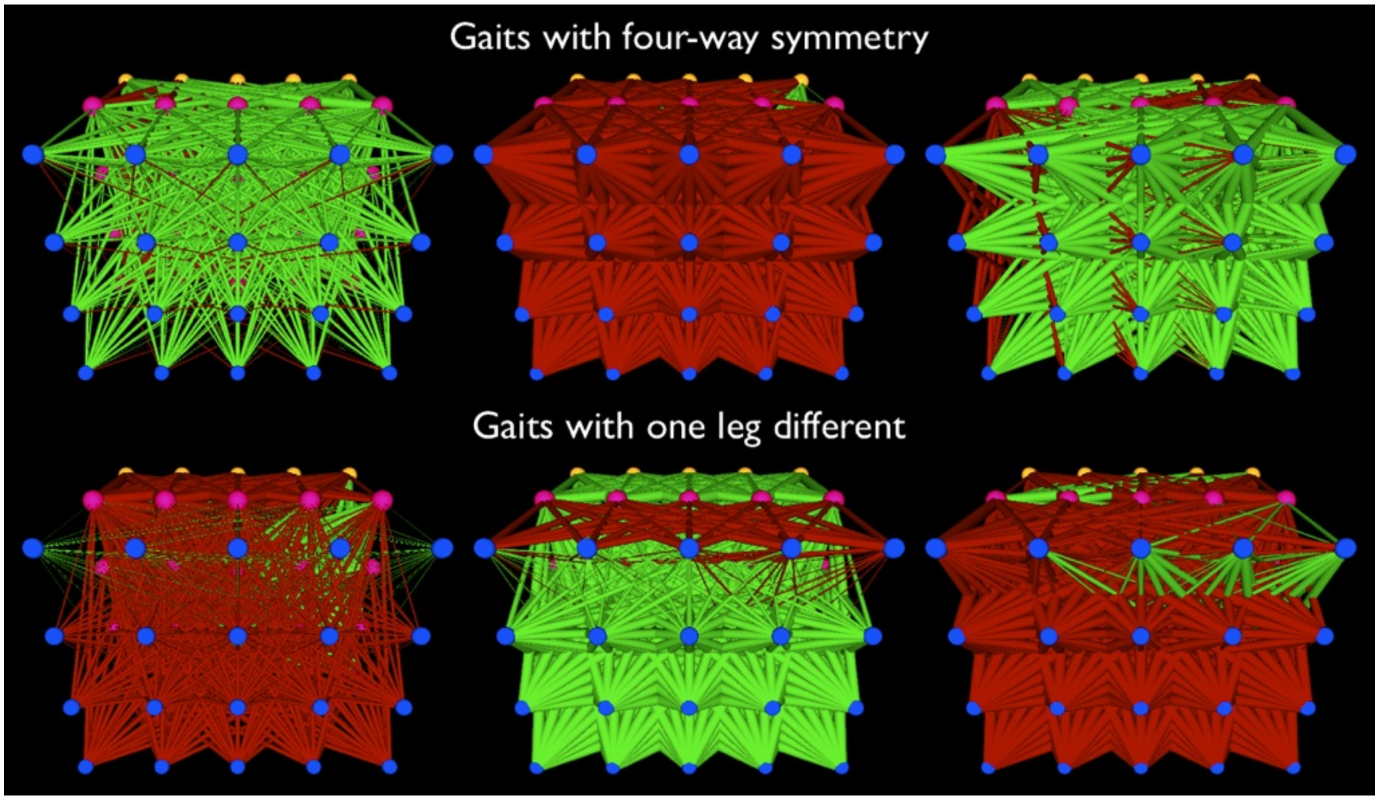

Fig. 17. Correlating ANN Regularities to Different Behaviors. It is possible to recognize ANN patterns that produce different robotic gaits. The ANNs in the top row all generate a four-way symmetric gait. The weight patterns in these ANNs appear similar for all rows (legs are controlled by separate rows of nodes). The ANNs in the bottom row have three legs moving together and one leg in anti-phase. That exception leg is controlled by the nodes in the top row, which have a different pattern of weights than the other three rows. These views are from the back (looking at the outputs), as indicated by the color/shade scheme described in the caption of Fig. 15.

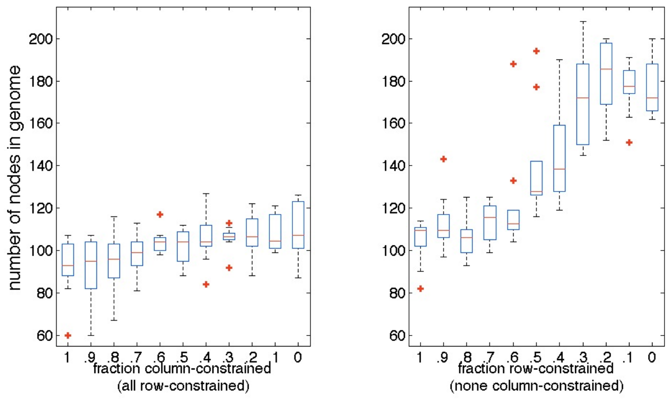

Fig. 18. Genome Size Increases with Problem Irregularity. The number of nodes in the CPPN genomes for HyperNEAT on the Bit Mirroring problem as irregularity is scaled from low (left) to high (right). The number of genomic nodes increases as the problem becomes more irregular.

We also performed this same analysis on data from a previous publication that created irregularity in the Quadruped Controller problem in a different way, by randomizing the geometric locations of each of the input and output nodes [8]. In the regular version of the problem, the nodes are laid out in a configuration that appears regular to a human engineer (e.g., the representation in Fig. 7). In the irregular version, the geometric locations of the nodes are randomized in each trial. We analyze data from three different experiments, where the nodes are represented with geometric coordinates in one, two, and three dimensions, respectively [8]. We previously reported that the performance of HyperNEAT is significantly higher in all three experiments on the regular version of the problem [8]. An analysis of the number of nodes in the CPPN genomes reveals that, for all three experiments, genome sizes are significantly smaller in the regular treatment than the irregular treatment ($p < 0.05$).

H. HyperNEAT is More Evolvable

One of the touted benefits of indirect encodings is that the reuse of genetic information that produces regularity also enables coordinated mutational effects, which can be beneficial [5], [22]. It has previously been shown that a different indirect encoding based on L-systems produces more beneficial mutations than a direct encoding control [22]. This section investigates whether HyperNEAT similarly tends to produce a higher distribution of fitness values in mutated offspring than its direct encoding controls.

We analyzed the difference in fitness between organisms and their offspring in the Quadruped Controller problem in all cases where offspring were produced solely by mutation. While the majority of organisms were produced by crossover and mutation, this analysis isolates the impact of mutational effects. Over 1.3 million, 1.7 million, and 1.5 million organisms

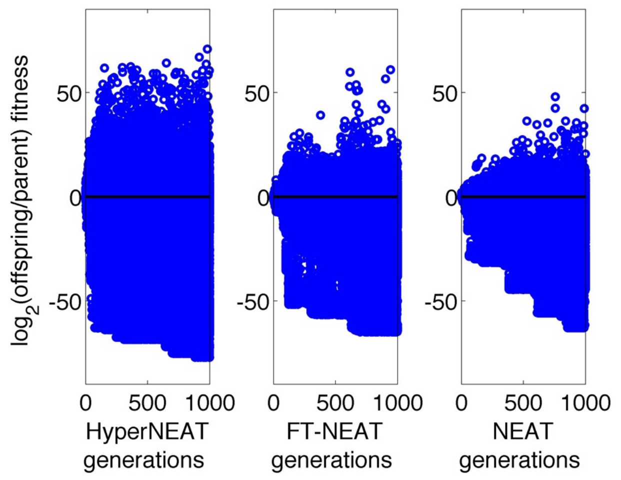

Fig. 19. The Fitness Changes Caused by Mutations with Different Encodings. Each circle represents the ratio of parent fitness over offspring fitness. Positive values indicate an offspring that is more fit than its parents, and higher numbers indicate larger fitness improvements. The inverse is true for negative numbers.

were produced solely via mutation for HyperNEAT, FT-NEAT, and NEAT treatments, respectively, providing a substantial sample size.

Overall, the indirect encoding HyperNEAT produces a wider *range* of fitness changes than the direct encodings (Fig. 19). HyperNEAT also has a distribution of fitness values with a higher median than both FT-NEAT and NEAT ($p < 0.001$). While HyperNEAT also produces more extreme negative fitness changes, they are balanced by more extreme positive fitness changes. For example, with respect to the ratio of parent fitness to offspring fitness, 5.8% of HyperNEAT offspring have a positive value greater than 20, whereas for FT-NEAT and NEAT only 0.35% and 0.21% do, respectively (Fig. 19). Despite the many extreme negative fitness changes, it appears that the continuous production of organisms that are much more fit than their parents fuels the success of HyperNEAT over FT-NEAT and NEAT on this problem.

I. HyperNEAT's Bias toward Regularity Hurts its Performance on Problems with Irregularity, as Demonstrated by a New Algorithm Called HybrID

The previous sections have documented that HyperNEAT's performance decreases as the irregularity of a problem increases. One explanation is that HyperNEAT's bias toward producing regular solutions makes it less likely to create the phenotypic irregularities necessary to account for problem-irregularities. The clearest example of this phenomenon is on the Target Weights problem in the treatment in which 90% of the weights had the same value, but 10% of the weights had different randomly-assigned values (Fig. 8). In a few generations, HyperNEAT discovers and exploits the regularity of the problem by setting 100% of its weights to the value that is correct for 90% of them. For the remaining hundreds of generations in the experiment, however, HyperNEAT fails to make exceptions to that regular pattern to account for the 10% of irregular link values. While the patterns observed in visualizations of the ANNs produced by HyperNEAT demonstrate that HyperNEAT can in fact produce some variation on overall patterns, many of those exceptions are themselves regular and affect whole geometric regions (Section IV-E). The Target Weights problem shows that changing specific weights to match an irregular target is challenging for HyperNEAT. However, such fine-tuning of neural connections may be required for real-world problems that have some degree of irregularity.

That HyperNEAT's performance generally decreases with problem irregularity suggests the hypothesis that HyperNEAT would perform better on such problems if it were able to both generate regularities and irregularities. An alternate explanation for the performance drop is that the less-regular problems are simply harder. One way to test whether the first hypothesis is correct is to create an algorithm that can generate regularities and then adjust those regularities to create irregularities.

Because indirect encodings excel at producing regular patterns, and direct encodings excel at producing irregular patterns, the combination of the two may produce both. The Hybridization of Indirect and Direct encodings (HybrID) algorithm [6] is based on this idea (Fig. 20).

While we are not aware of any prior work that specifically combines direct and indirect encodings, researchers have previously altered representations during evolutionary search, primarily to change the precision of values being evolved by genetic algorithms [12]. Other researchers have employed non-evolutionary optimization techniques to fine-tune the details of evolved solutions [18]. However, such techniques do not leverage the benefits of indirect encodings.

While the name HybrID applies to any combination of indirect and direct encodings, this paper reports results for one specific implementation called a *switch-HybrID* [6], wherein an indirect encoding is run first and then a switch is made to a direct encoding. Specifically, HyperNEAT is the

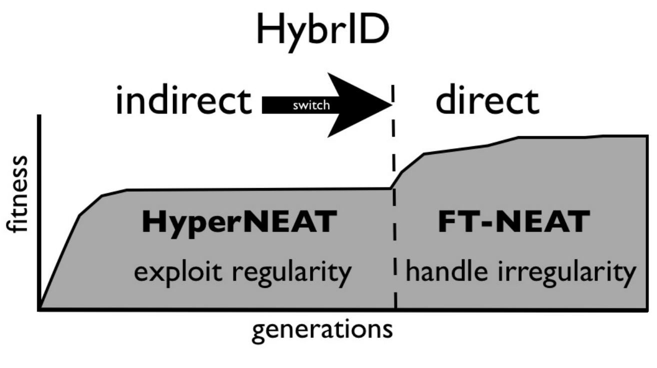

Fig. 20. Hybridizing Indirect and Direct Encodings in the HybrID Algorithm. The HybrID implementation in this paper evolves with HyperNEAT in the first phase until a switch is made to FT-NEAT. The idea is that the indirect encoding phase can produce regular weight patterns that can exploit problem regularity, and the direct encoding phase can fine tune that pattern to account for problem irregularities. In this hypothetical example, large fitness gains are initially made by the indirect encoding because it exploits problem regularity, but improvement slows because the indirect encoding cannot adjust its regular patterns to handle irregularities in the problem. Fitness increases again, however, once the direct encoding begins to fine-tune the regular structure produced by the indirect encoding.

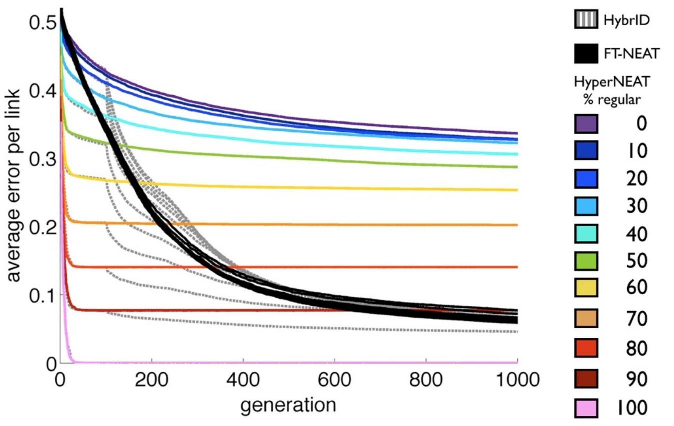

Fig. 21. A Comparison of HyperNEAT, FT-NEAT, and HybrID on a Range of Problem Regularities for the Target Weights Problem. For each regularity level, a HybrID line (dashed gray) departs from the corresponding HyperNEAT line (colored) at the switch point (generation 100). The performance of FT-NEAT (black lines) is unaffected by the regularity of the problem, which is why the lines are overlaid and indistinguishable. HybrID outperforms HyperNEAT and FT-NEAT in early generations on versions of the problem that are mostly regular but have some irregularities.

encoding for each generation until the switch point, when each HyperNEAT ANN phenotype is converted into an FT-NEAT genome. Evolution then continues as normal with FT-NEAT until the end of the experiment. HyperNEAT and FT-NEAT serve as the indirect and direct encodings in the HybrID implementation in this paper to provide fair comparisons to the results from previous sections. For each of the HybrID experiments in this paper, a different switch point is chosen for illustrative purposes. In future applications of the HybrID algorithm, it may be more effective to choose a separate switch point for each run automatically based on the rate of fitness improvement (or lack thereof).

1) Target Weights: HybrID's performance on the Target Weights problem reveals that it does combine the regularity-generating attribute of indirect encodings with the irregularity-generating attribute of direct encodings (Fig. 21). At the switch point of 100 generations, HybrID immediately makes noticeable gains over HyperNEAT, and 150 generations later these gains are significant on all treatments except the perfectly regular one ($p < 0.001$). This result confirms the hypothesis that HyperNEAT can do better on some irregular problems if a further process of refinement creates some irregularity within its regular patterns. It is additionally interesting that HybrID significantly outperforms the direct encoding FT-NEAT on regular versions of the problem early in the experiment ($p < 0.01$ at generation 250 on the 70%, 80% and 90% regular problems); The HyperNEAT phase of HybrID first discovers the regularity of the problem, giving the FT-NEAT phase of HybrID a head start over FT-NEAT on these problems. While the performance of HybrID and FT-NEAT is similar by the end of the experiment, that result is likely because this problem is simple and has no interactions (epistasis) between individual link weight values. Direct encodings are expected to perform well on problems without epistasis, but most real world problems are highly epistatic.

2) Bit Mirroring: On the Bit Mirroring problem, which does have epistasis, HybrID's performance ties HyperNEAT's performance on regular versions of the problem, and significantly outperforms HyperNEAT on problems with a certain level of irregularity (Fig. 22; see figure for statistical significance per treatment). This result further highlights that HyperNEAT produces regular patterns that can benefit from a refining process that generates irregularity. The performance gap between HybrID and HyperNEAT is largest on problems with intermediate levels of regularity. The gap in performance narrows on the most irregular treatments because such configurations are difficult and both algorithms perform poorly.

These data are pooled across 10 runs per treatment on a 7×7 grid. Each experiment lasted 5000 generations with a switch point at 2500 generations. A comparison to FT-NEAT is not shown because HyperNEAT outperforms FT-NEAT on all versions of this problem (Section IV-A2).

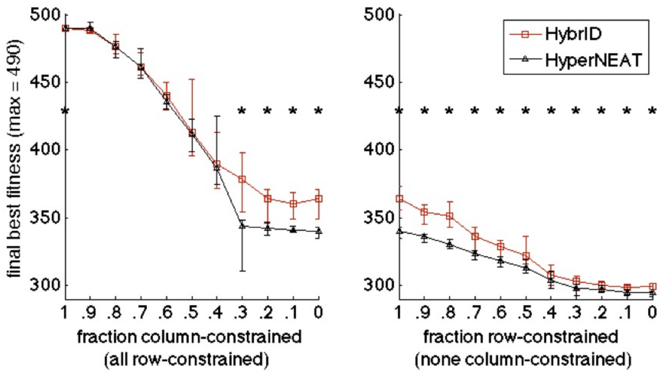

Fig. 22. The Performance of HybrID Vs. HyperNEAT on the Bit Mirroring Problem. Regularity decreases from left to right. Plotted are median values ± the 25th and 75th quartiles. Asterisks indicate $p < 0.05$.

3) Quadruped Controller: HybrID outperforms HyperNEAT on every version of the Quadruped Controller problem (Fig. 23), although the difference is significant only on problems with a certain amount of irregularity ($p < 0.01$ on treatments with four or more faulty joints). HybrID increases performance over HyperNEAT by 5%, 10%, 27%, 64%, and 44%, respectively, for the treatments with 0, 1, 4, 8, and 12 faulty joints. These substantial performance improvements on the Quadruped Controller problem, which is a challenging engineering problem, highlight the degree to which HyperNEAT's inability to produce irregularity on its own can harm its performance.

HybrID also outperforms both direct encodings on all treatments of the problem ($p < 0.05$). HyperNEAT significantly outperforms both direct encodings only on the two most regular versions of the problem ($p < 0.01$). That HybrID outperforms the direct encodings on irregular problems underscores that it does not just act like a direct encoding on irregular problems, but instead first leverages the indirect encoding's ability to exploit available regularities and then improves upon those by accounting for problem irregularities via the direct encoding.

It is instructive to examine how the FT-NEAT phase of

HybrID changes the patterns provided to it by the HyperNEAT phase. Visualizations of ANNs at the end of each HyperNEAT phase and the ANN for that same run after the FT-NEAT phase can provide clues to how HybrID generates its performance improvements. Examples from runs in the treatment with one faulty joint are shown in Fig. 24. In all cases, the FT-NEAT phase of HybrID makes no major changes to the overall regular pattern produced by the HyperNEAT phase (visualizations of ANNs after the HyperNEAT and FT-NEAT phases for each HybrID run are available at http://devolab.msu.edu/SupportDocs/Regularity). Evolution thus maintains the regular pattern HyperNEAT generates even while that pattern is being fine-tuned by the direct encoding.

The types of exceptions HybrID produces are different from those seen by HyperNEAT alone. In many cases, only a few weights are noticeably changed by the FT-NEAT phase of HybrID, and these changes occur in an irregular distribution. For example, in the run depicted in the left column of Fig. 24, HyperNEAT produces the regular pattern of inhibitory connections to all of the output nodes. FT-NEAT switches some of those to excitatory connections, which may have been difficult for HyperNEAT to do without changing many other weights. In another example run, depicted in the middle column of Fig. 24, the only noticeable change FT-NEAT made is the creation of a single, strong, excitatory connection. Of course, in both cases there are subtle changes in many of the link weights that do not stand out to the human eye.

In a third example run, changes were made during the FT-NEAT phase to many different weights, yet the overall patterns remained (Fig. 24, right column). Many of these changes are irregular, such as the weights switched from excitatory to inhibitory and vice versa in the top-left node, and the few links that switch to excitatory in the bottom row. What is unusual and fascinating about this example run, however, is that the direct encoding makes many *regular* changes. For example, most of the links in the top row proportionally increase in strength, which preserves the regular patterns. These visualizations demonstrate a rare case of a direct encoding producing coordinated phenotypic changes. It might be the case that the indirect encoding discovered regularities that put this organism on the side of a hill in a fitness landscape, but climbing that hill is difficult for the indirect encoding because mutations to the genome that increase the strength of connections in these nodes may change other weights and thus decrease fitness overall. The direct encoding does not have such constraints, and thus can increase the magnitude of all of these links. This hypothetical explanation illustrates how evolution can produce coordinated change through direct encodings if it starts in a place in the fitness landscape where there is a positive fitness gradient in the same direction for many link values. Interestingly, it is unlikely that the direct encoding would have discovered this starting point without the indirect encoding.

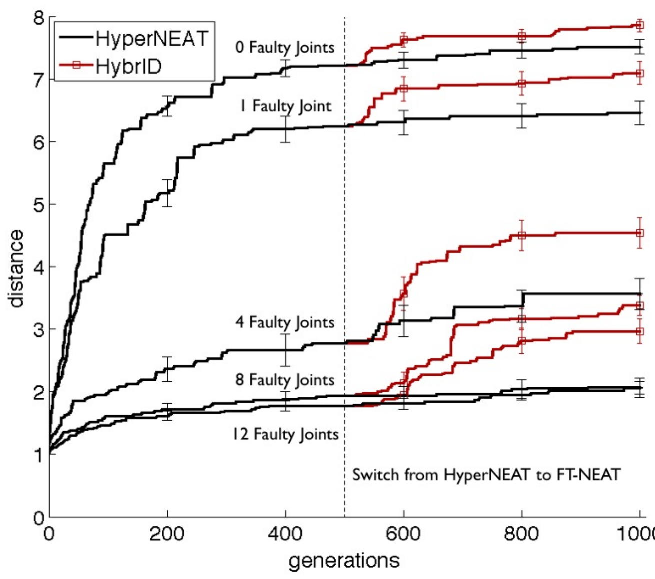

Fig. 23. The Performance of HybrID Vs. HyperNEAT on the Quadruped Controller Problem. Error bars show one standard error of the median. HybrID outperforms HyperNEAT on all versions of the Quadruped Controller problem. The increase generally correlates with the number of faulty joints.

V. DISCUSSION

While it is well-established that indirect encodings can outperform direct encodings on regular problems [10], [15], [20], [21], [23], [30], [34], [44], [46], no prior studies have investigated how indirect encodings compare to direct encodings on problems as regularity scales from high to low. Given that most realistic problems will not be at the extreme regular end of this continuum, it is important to understand how indirect encodings perform on problems with different levels of regularity.

On three different problems we have shown that indirect encodings are able to automatically exploit intermediate amounts of problem regularity, and that their performance improves as the regularity of the problem increases. Moreover, the indirect encodings outperform direct encoding controls on regular problems. We have also shed light on why indirect encodings perform better, which is because they produce both regular ANNs and regular behaviors that exploit problem regularity. These results suggests that indirect encodings may be an attractive alternative to direct encodings as evolutionary algorithms are applied to increasingly complicated engineering problems, because such problems are likely to contain regularities. The results from the Bit Mirroring problem also reveal that HyperNEAT is able to independently exploit different types of regularity within the same problem; scaling three different regularities (within-row, within-column, and inherent) independently contributes to overall performance [7]. One interesting caveat we discovered on all three problems, however, is that HyperNEAT does not exploit a particular type of regularity until the level of regularity within that type is above a threshold. Tests on additional problems and with different indirect encodings are required to discover how general this finding is. Additionally, further quantitative analyses of regularity in both evolved solutions and in the problems themselves would shed additional light on how general the findings in this paper are.

It is particularly noteworthy that HyperNEAT is able to automatically exploit the regularity of the challenging Quadruped

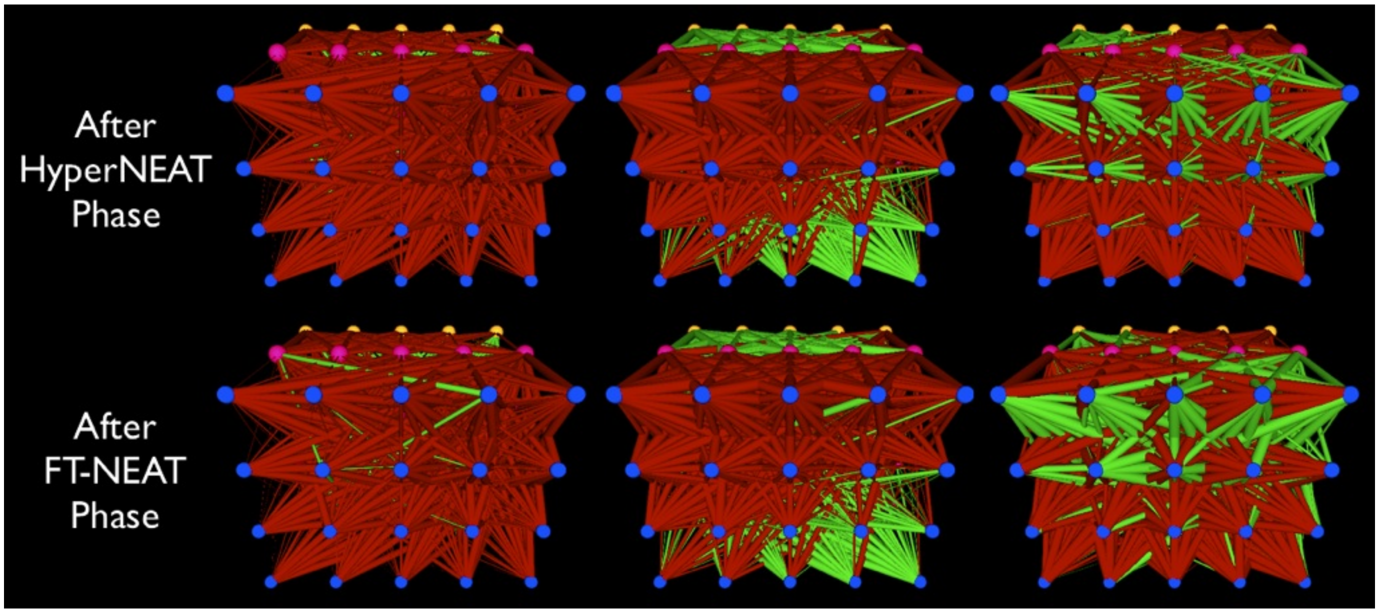

Fig. 24. Visualizations of the ANNs Produced at the End of the HyperNEAT Phase and FT-NEAT Phase of HybrID for Three Example Runs.

Controller problem. This result is important because evolutionary algorithms have previously performed poorly when evolving gaits for legged robots because they could not automatically exploit the problem's regularities: to perform well, researchers needed to manually identify such regularities and force the encoding to exploit them [1], [13], [24], [25], [33], [50], [51]. This manual approach is time consuming, restrictive, and may fail to identify helpful regularities.

Because indirect encodings are biased toward producing regular patterns in solutions, it is important to understand the degree to which they can vary and make exceptions to these patterns. Two separate lines of evidence confirm that HyperNEAT has difficulty making certain types of exceptions. Initially, its performance decreases as a problem becomes more irregular. Second, the HybrID algorithm boosts performance over HyperNEAT on intermediately regular problems, confirming that HyperNEAT is not creating some irregularities that would aid performance. On the other hand, visualizations of HyperNEAT ANNs reveal that HyperNEAT can create variations on patterns, and even exceptions for single nodes. However, these variations and exceptions themselves are regular, suggesting that HyperNEAT creates its exceptions by adding one regularity to another, with the result being an overall regularity with a regular variation within it. The HyperNEAT visualizations rarely demonstrate cases where single weights violate the prevailing pattern. HybrID, on the other hand, does provide examples of such single-link exceptions. The significant boost in performance by HybrID implies that the ability to make such radical exceptions at the single-link level is sometimes important.

HybrID's performance increase over HyperNEAT, combined with investigations into the different types of exceptions HybrID and HyperNEAT make, suggest that HyperNEAT can benefit from a process of refinement that adjusts individual link patterns in an irregular way. While a direct encoding provides such refinement in this paper, there are other candidate refinement processes. One intriguing possibility is that lifetime adaptation via learning can play a similar role [31], [36], [40], [41]. Lifetime learning algorithms could adjust the overall regular patterns produced by HyperNEAT to account for necessary irregularities. Having a learning algorithm serve as the refining process may be superior to a direct encoding, especially as ANNs scale closer to the size of brains in nature. While HybrID works well on the problems in this paper, its direct encoding component may ultimately encounter the same scaling challenges that all direct encodings face on high-dimensional problems.

The ANN weight patterns produced by HyperNEAT alone are also interesting because they demonstrate the types of regularities that can be produced by an indirect encoding that incorporates geometric concepts from developmental biology. The pictures evolved with CPPNs reveal that CPPNs can encode many features observed in natural animal bodies, such as symmetry and serial repetition, with and without variation (Fig. 2) [38]. The visualizations presented here of ANNs evolved with CPPNs demonstrate that HyperNEAT can generate these same properties in ANNs. It is clear by looking at the different visualizations that different geometric patterns are being created and combined to produce complex neural patterns, which is reminiscent of how nature produces complex ANNs and bodies [2].

Outside the field of neuroevolution, other techniques have improved performance by biasing neural networks towards regular weight patterns, such as weight sharing [27], [37] and convolutional networks [26]. However, such methods typically involve the researcher imposing a certain regularity on the ANN (such as a subset of weights all being identical). The fact that these techniques were successful demonstrates that regular patterns in neural connections can be beneficial. However, these methods typically do not automatically discover which regularities to create. HyperNEAT is novel because it explores a large space of possible geometric regularities to find those that enhance performance. This ability to discover and encode different regularities suggests that HyperNEAT can potentially adapt the regularities it produces to different domains, instead of needing to be manually tuned for each domain.

VI. CONCLUSION

This paper contains the first extensive study in the field of evolutionary computation comparing an indirect encoding to direct encoding controls across a continuum of problems with different levels of regularity. On three different problems the performance of the indirect encoding improved with the regularity of the problem, and the indirect encoding outperformed the direct encodings on more regular versions of problems. The indirect encoding was able to exploit problem regularity by generating regular neural networks that produced regular behaviors. The specific indirect encoding studied is based on concepts from developmental biology involving geometric patterning, which enabled domain knowledge and preferences to be injected into the algorithm. Moreover, this generation and combination of geometric coordinate frames created regular weight patterns in neural networks that are visually complex and resemble regularities seen in natural brains.

The indirect encoding's bias toward regularity hurt its performance on problems that contained some irregularity. A new algorithm that first evolves with an indirect encoding and then switches to a direct encoding was able to outperform the indirect encoding alone. This HybrID algorithm outperformed the indirect encoding because it made subtle adjustments to regular patterns to account for problem irregularities. The success of this approach suggests that indirect encodings may be most effective not as stand-alone algorithms, but in combination with a refining process that adjusts regular patterns in irregular ways to account for problem irregularities.

Overall, this paper provides a more comprehensive picture of how indirect encodings compare to direct encodings by evaluating them across a continuum from high to low problem regularity. The paper also suggests a path forward that combines the pattern-producing power of indirect encodings with a process of refinement to account for the irregularities that are likely to exist in challenging problems.

ACKNOWLEDGMENT

The authors would like to thank the members of the Digital Evolution Laboratory at Michigan State University, in particular Philip K. McKinley, Benjamin E. Beckmann, and David Knoester.

REFERENCES

[1] R.D. Beer and J.C. Gallagher. Evolving dynamical neural networks for adaptive behavior. *Adaptive behavior*, 1(1):91, 1992.

[2] S.B. Carroll. *Endless forms most beautiful: The new science of evo devo and the making of the animal kingdom*. WW Norton & Company, 2005.

[3] S.B. Carroll, J.K. Grenier, and S.D. Weatherbee. *From DNA to diversity*. Blackwell Science Malden, MA, 2001.

[4] J. Clune, B.E. Beckmann, P.K. McKinley, and C. Ofria. Investigating whether HyperNEAT produces modular neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 635–642. ACM, 2010.

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

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

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

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

[9] D.B. D'Ambrosio, J. Lehman, S. Risi, and K.O. Stanley. Evolving policy geometry for scalable multiagent learning. In *Proceedings of the Conference on Autonomous Agents and Multiagent Systems*, pages 731–738, 2010.

[10] D.B. D'Ambrosio and K.O. Stanley. A novel generative encoding for exploiting neural network sensor and output geometry. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 974–981. ACM, 2007.

[11] D.B. D'Ambrosio and K.O. Stanley. Generative encoding for multiagent learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 819–826. ACM, 2008.

[12] A. Eiben, Z. Michalewicz, M. Schoenauer, and J. Smith. Parameter control in evolutionary algorithms. *Parameter Setting in Evolutionary Algorithms*, pages 19–46, 2007.

[13] D. Filliat, J. Kodjabachian, and J.A. Meyer. Evolution of neural controllers for locomotion and obstacle avoidance in a six-legged robot. *Connection Science*, 11(3):225–242, 1999.

[14] J.C. Gallagher, R.D. Beer, K.S. Espenschied, and R.D. Quinn. Application of evolved locomotion controllers to a hexapod robot. *Robotics and Autonomous Systems*, 19(1):95–103, 1996.

[15] J. Gauci and K. Stanley. Generating large-scale neural networks through discovering geometric regularities. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 997–1004. ACM, 2007.

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

[17] J. Gauci and K.O. Stanley. Autonomous evolution of topographic regularities in artificial neural networks. *Neural Computation*, 22(7):1860–1898, 2010.

[18] J.B. Grimbleby. Automatic analogue circuit synthesis using genetic algorithms. *IEE Proceedings Circuits Devices and Systems*, 147(6):319–324, 2000.

[19] F. Gruau. Automatic definition of modular neural networks. *Adaptive Behavior*, 3(2):151–183, 1994.

[20] F. Gruau and E.N.S.L. LIP. Genetic synthesis of Boolean neural networks with a cell rewriting developmental process. In *Combinations of Genetic Algorithms and Neural Networks, International Workshop on*, pages 55–74, 1992.

[21] F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Proceedings of the First Annual Conference on Genetic Programming*, pages 81–89. MIT Press, 1996.

[22] G.S. Hornby, H. Lipson, and J.B. Pollack. Generative representations for the automated design of modular physical robots. *IEEE Transactions on Robotics and Automation*, 19(4):703–719, 2003.

[23] G.S. Hornby and J.B. Pollack. Creating high-level components with a generative representation for body-brain evolution. *Artificial Life*, 8(3):223–246, 2002.

[24] G.S. Hornby, S. Takamura, T. Yamamoto, and M. Fujita. Autonomous evolution of dynamic gaits with two quadruped robots. *IEEE Transactions on Robotics*, 21(3):402–410, 2005.

[25] J. Kodjabachian and J.A. Meyer. Evolution and development of neural controllers for locomotion, gradient-following, and obstacle-avoidance in artificial insects. *IEEE Transactions on Neural Networks*, 9(5):796–812, 1998.

[26] S. Lawrence, C.L. Giles, A.C. Tsoi, and A.D. Back. Face recognition: A convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, 1997.

[27] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[28] H. Lipson. Principles of modularity, regularity, and hierarchy for scalable systems. *Journal of Biological Physics and Chemistry*, 7(4):125, 2007.

[29] H. Liu and H. Iba. A hierarchical approach for adaptive humanoid robot control. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2004.

[30] J.F. Miller. Evolving a self-repairing, self-regulating, French flag organism. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 129–139. Springer, 2004.

[31] S. Nolfi and D. Floreano. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press, Cambridge, MA, 2000.

[32] R.A. Raff and J. Slack. *The shape of life: genes, development, and the evolution of animal form*. University of Chicago Press Chicago, 1996.

[33] M. Raibert, M. Chepponis, and H. Brown Jr. Running on four legs as though they were one. *IEEE Journal of Robotics and Automation*, 2(2):70–82, 1986.

[34] J. Reisinger and R. Miikkulainen. Acquiring evolvability through adaptive representations. In *Proceedings of the Genetic and Evolutionary Computation Conference*, page 1052. ACM, 2007.

[35] C. Ridderstrom. Legged locomotion control—a literature survey. Technical Report TRITA-MMK, Royal Institute of Technology, Stockholm, Sweden, ISSN 1400-1179. 1999.

[36] S. Risi and K.O. Stanley. Indirectly encoding neural plasticity as a pattern of local rules. *Proceedings of the 11th International Conference on Simulation of Adaptive Behavior*, To Appear, 2010.

[37] D.E. Rumelhart, GE Hinton, and R.J. Williams. *Learning internal representations by error propagation, Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*. MIT Press, Cambridge, MA, 1986.

[38] J. Secretan, N. Beato, D.B. D Ambrosio, A. Rodriguez, A. Campbell, and K.O. Stanley. Picbreeder: evolving pictures collaboratively online. In *Proceedings of the Computer Human Interaction Conference*, pages 1759–1768. ACM, 2008.

[39] K. Sims. Evolving 3D morphology and behavior by competition. *Artificial Life*, 1(4):353–372, 1994.

[40] A. Soltoggio, J.A. Bullinaria, C. Mattiussi, P. Durr, and D. Floreano. Evolutionary Advantages of Neuromodulated Plasticity in Dynamic, Reward-based Scenarios. *Artificial Life*, 11:569, 2008.

[41] A. Soltoggio, P. Durr, C. Mattiussi, and D. Floreano. Evolving neuromodulatory topologies for reinforcement learning-like problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 2007. Citeseer, 2007.

[42] C. Southan. Has the yo-yo stopped? An assessment of human protein-coding gene number. *Proteomics*, 4(6):1712–1726, 2004.

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

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

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

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

[47] K.O. Stanley and R. Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21(1):63–100, 2004.

[48] G.F. Striedter. *Principles of brain evolution*. Sinauer Associates Sunderland, MA, 2005.

[49] M.E. Taylor, P. Stone, and Y. Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1):2125–2167, 2007.

[50] R. Téllez, C. Angulo, and D. Pardo. Evolving the walking behaviour of a 12 dof quadruped using a distributed neural architecture. *Biologically Inspired Approaches to Advanced Information Technology*, pages 5–19, 2006.

[51] V.K. Valsalam and R. Miikkulainen. Modular neuroevolution for multilegged locomotion. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 265–272. ACM, 2008.

[52] P. Verbancsics and K.O. Stanley. Task transfer through indirect encoding. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2010 (to appear).

[53] D. Wettergreen and C. Thorpe. Gait generation for legged robots. In *IEEE International Conference on Intelligent Robots and Systems*, pages 1413–1420, 1992.

[54] K. Wolff and P. Nordin. An evolutionary based approach for control programming of humanoids. In *Proceedings of the 3rd International Conference on Humanoid Robots*, 2003.

Jeff Clune is a Postdoctoral Fellow in Hod Lipson's lab at Cornell University, funded by a Postdoctoral Research Fellowship in Biology from the National Science Foundation. He studies generative encodings, which enhance evolutionary algorithms by augmenting them with concepts from developmental biology. Such concepts enable the assembly of complex forms from compact genomes. He combines these generative encodings with neuroevolution, a technology that harnesses the power of evolution to construct artificial neural networks (ANNs). Evolving ANNs with generative encodings creates large-scale, structurally organized ANNs that produce sophisticated, coordinated behaviors. Jeff demonstrates the capabilities of such ANNs in robotic control problems. He also develops evolutionary algorithms to investigate open questions in evolutionary biology, and has published work on the evolution of altruism, phenotypic plasticity, and evolvability. This work contributes to biology, because it improves our knowledge of evolving systems, and enhances computer science, because it helps design better evolutionary algorithms. Jeff is the co-chair of the Generative and Developmental Systems track at GECCO, the Genetic and Evolutionary Computation Conference (2010-2011). He won the best paper award in that track in 2009. He has a Ph.D. in computer science from Michigan State University, a master's degree in philosophy from Michigan State University, and a bachelor's degree in philosophy from the University of Michigan. Jeff is also a co-author of Avida-ED, a software tool for teaching evolution. Articles about his research have appeared in many news publications, including New Scientist, the Daily Telegraph, Slashdot, and U.S. News & World Report.

Kenneth O. Stanley is an assistant professor in the School of Electrical Engineering and Computer Science at the University of Central Florida. He received a B.S.E. from the University of Pennsylvania in 1997 and received a Ph.D. in 2004 from the University of Texas at Austin. He is an inventor of the Neuroevolution of Augmenting Topologies (NEAT) and HyperNEAT algorithms for evolving complex artificial neural networks. His main research contributions are in neuroevolution (i.e. evolving neural networks), generative and developmental systems, coevolution, machine learning for video games, and interactive evolution. He has won separate best paper awards for his work on NEAT, NERO, NEAT Drummer, HyperNEAT, novelty search, and Galactic Arms Race. He is the chair of the IEEE Task Force on Computational Intelligence and Video Games, an associate editor of IEEE Transactions on Computational Intelligence and AI in Games, and chaired the Generative and Developmental Systems track at GECCO from 2007 to 2009. He recently joined the editorial board of the Evolutionary Computation journal.

Robert T. Pennock is Professor at Michigan State University, where he is on the faculty of the Lyman Briggs College, the Philosophy Department, the Department of Computer Science, and the Ecology, Evolutionary Biology and Behavior graduate program. He is one of the leads of the BEACON Center for the Study of Evolution in Action. His scientific research in experimental evolution and evolutionary computation focuses especially on the emergence of complexity and intelligent behavior, and has been featured in Discover, New Scientist, Slashdot, and many other periodicals. His philosophical research interests are in philosophy of biology, artificial life, and in the relationship of epistemic and ethical values in science. Pennock speaks regularly around the country on these topics, and has been named a national Distinguished Lecturer by Sigma Xi, The Scientific Research Society. A national leader in evolution education, he was an expert witness in the historic Kitzmiller v. Dover Area School Board case that ruled that Intelligent Design creationism is not science, but sectarian religion, and that teaching it is the public schools is unconstitutional. His book Tower of Babel: The Evidence against the New Creationism has been reviewed in over fifty publications; the New York Review of Books called it the best book on creationism in all its guises. A Fellow of the American Association for the Advancement of Science, Pennock has served on the AAAS Committee on the Public Understanding of Science and Technology and the National Academies of Science Evolution, Creationism and Science authoring committee.

Charles Ofria is the director of the MSU Digital Evolution Laboratory, deputy director of the BEACON Center for the Study of Evolution in Action, and an associate professor in the Department of Computer Science and Engineering at Michigan State University. His research lies at the intersection of computer science and evolutionary biology, developing a two-way flow of ideas between the fields. In particular, he is focused on understanding the evolution of biological complexity and complex behaviors including evolvability, navigation, cooperation, division of labor, and intelligence. He received a bachelor's degree in 1994 from SUNY Stony Brook with a triple major in pure math, applied math, and computer science. In 1999, he received a Ph.D. from the California Institute of Technology in Computation and Neural Systems, followed by a postdoc in the Center for Microbial Ecology at MSU. Dr. Ofria is the architect of the Avida Digital Evolution Research Platform, which is downloaded over a thousand times per month for use in research and education at dozens of universities around the world.