

How a Generative Encoding Fares as Problem-Regularity Decreases

Jeff Clune¹, Charles Ofria¹, and Robert T. Pennock^{1,2}

¹ Department of Computer Science and Engineering,

² Department of Philosophy & Lyman Briggs College

Michigan State University, East Lansing, MI, USA 48824

{jclune, ofria, pennock5}@msu.edu

Abstract. It has been shown that generative representations, which allow the reuse of code, perform well on problems with high regularity (i.e. where a phenotypic motif must be repeated many times). To date, however, generative representations have not been tested on irregular problems. It is unknown how they will fare on problems with intermediate and low amounts of regularity. This paper compares a generative representation to a direct representation on problems that range from having multiple types of regularity to one that is completely irregular. As the regularity of the problem decreases, the performance of the generative representation degrades to, and then underperforms, the direct encoding. The degradation is not linear, however, yet tends to be consistent for different types of problem regularity. Furthermore, if the regularity of each type is sufficiently high, the generative encoding can simultaneously exploit different types of regularities.

Keywords: Evolution, regularity, modularity, ANN, NEAT, HyperNEAT.

1 Introduction

While the field of evolutionary computation has produced impressive results, the complexity of its evolved solutions pales in comparison to organisms in the natural world. One of several likely reasons for this difference is that evolutionary computation typically uses a *direct encoding* (also known as *direct representation*), where, relative to some environment E, every part of the phenotype is coded for separately in the genome. Given that natural organisms can contain trillions of parts (e.g. cells in the human body), a direct representation of such an organism would require a genome with at least that many separate genetic elements. We do not find such inefficient genomes in nature. An alternative is a *generative encoding* (or *generative representation*), where, relative to E, elements in a genome can be reused to produce many parts of a phenotype [1-9]. For example, about 25,000 genes encode the information that produces the trillions of parts that make up a human [10]. Generative encodings allow evolution to search a genotype space with far fewer dimensions than that of the final phenotype. Further benefits of generative encodings are that the reuse of code facilitates the evolution of modular phenotypes and that mutations can produce coordinated phenotypic effects (e.g. one mutation lengthening all legs by the same amount).

Previous researchers have found that generative encodings produce more modular, complex phenotypes, higher fitnesses, and more beneficial mutations on average than direct encoding controls [1].

While it has been shown that generative encodings can outperform direct encodings [1-7], in every case the authors of this paper are aware of, the problem was nearly perfectly regular or the regularity of the problem was unspecified and ambiguous. Gruau's work evolving neural nets with the generative encoding called Cellular Encoding used problem domains of bit parity or bit symmetry [4], which are both highly regular, or pole-balancing [5], where the regularity of the problem is unknown. Even for the pole-balancing problem, however, there is left-right symmetry and just a few tasks (e.g. calculating velocity) need to be repeated many times. Hornby [1] demonstrated that a generative encoding based on L-systems outperformed a direct encoding control when applied to the perfectly regular parity problem and to evolving tables and mobile creatures (where repeating similar leg modules gave huge fitness gains). The regularity of the Nothello game from [7] is unknown. Recently, a new generative encoding, called Hypercube-based NEAT (HyperNEAT), has been shown to outcompete a direct encoding control on two problems that require the repetition of the same network motif [2,3].

These works show that generative encodings do well on highly regular problems, but they raise the question of whether generative encodings achieve their increased performance in regular problem domains at the cost of performing poorly in irregular problem domains. For example, how good are generative encodings at producing an exception to the rule? Furthermore, do they provide advantages for low and intermediate amounts of regularity, or only when regularity is high? What is needed are tests of generative versus direct encodings on problems that allow us to explicitly vary *only* their regularity. Such investigations are conducted in this paper.

2 The Experimental System

This study uses a generative encoding that evolves neural nets, which is one of the common uses for generative encodings within the field of evolutionary computation [1-8]. HyperNEAT [2,3] was recently introduced as a generative representation that can evolve neural nets using the principles of the widely used NeuroEvolution of Augmenting Topologies (NEAT) algorithm [11]. HyperNEAT evolves Compositional Pattern Producing Networks (CPPNs), each of which is a function that takes an input and produces an output. The inputs to the CPPN function are a constant bias value and the locations on a Cartesian grid of both an input node (e.g. $\langle x_1=4, y_1=4 \rangle$) and an output node (e.g. $\langle x_2=5, y_2=5 \rangle$). The function takes these five values (bias, x_1 , y_1 , x_2 , y_2) as input and produces an output value that determines the weight of the link between the input and output node. All pairwise combinations of input and output nodes are iteratively passed as inputs to a given CPPN to determine what the weight value is between each input node and each output node. Thus the CPPN function is a genome that encodes for a neural network phenotype.

Evolution proceeds in HyperNEAT by evolving a population of CPPN functions. Each CPPN is itself a directed graph network where each node is a math function comprised of the following functions: sine, sigmoid, cosine, Gaussian, square,

absolute root, linear, or one's complement. The nature of the functions used can create a wide variety of desirable properties, such as symmetry (e.g. an absolute value or Gaussian function) and repetition (e.g. a sine or cosine function) that evolution can take advantage of. That a directed network of functions is used allows nested coordinate frames to develop, such that, for instance, a sine function used early in the network can create a repeating theme that, when passed into the symmetrical absolute value function, creates a repeating series of symmetrical motifs. This is similar to how natural organisms develop. For example, many organisms set up a repeating coordinate frame (e.g. body segments) within which are symmetrical coordinate frames (e.g. left-right body symmetry). The links between each node in a CPPN have a weight value that can magnify or diminish the values that pass along them. The ability to change these weights enables evolution to, for example, give strong weight to one part of the network generating symmetry while rendering the influence of another aspect of the network more subtle. When CPPNs are evolved artificially with humans doing the selection, the evolved shapes look surprisingly beautiful, complex and natural [9]. More importantly, they exhibit the desirable features of generative encodings, namely, the repetition of themes, symmetries and hierarchies, with and without variation.

Variation in HyperNEAT occurs when mutations change the CPPN function networks. Mutations can add a node to the graph, which results in the addition of a function to the network, or change the weights of links within the network. The evolution of the population of CPPN networks occurs according to the principles of NEAT, which was originally designed to evolve neural nets. NEAT can be fruitfully applied to CPPNs because the population of CPPN networks is similar in structure to a population of neural networks.

The NEAT algorithm is unique in three main ways [11]. Initially, it starts with small genomes that encode simple networks and slowly 'complexifies' them via mutations that add nodes and links to the network. This complexification enables the algorithm to evolve the network topology in addition to its weights. Secondly, it uses a fitness sharing mechanism that preserves diversity in the system 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, it uses historical information to perform crossover in a way that is effective yet avoids the need for expensive topological analysis. A full explanation of HyperNEAT [2, 3] and NEAT [11] can be found elsewhere.

It is helpful that a good direct encoding version of NEAT exists that can serve as a control. Perception NEAT (P-NEAT), so named because it evolves a series of perceptrons, has been previously used to compare the generative encoding of HyperNEAT with a direct encoding that is similar to HyperNEAT in all ways, save its use of the generative CPPNs [2, 3]. P-NEAT directly evolves the neural net phenotypes. It is the same as NEAT without the complexification. In other words, P-NEAT uses evolution to tune the weights of a network with a fixed topology. Since in HyperNEAT the complexification is performed on the CPPN, but the resultant neural network topology remains fixed, the topology of the P-NEAT neural network is also fixed. The rest of the elements from NEAT (e.g. fitness sharing) remain the same between HyperNEAT and P-NEAT, making the latter a good control.

Following Gauci and Stanley (2007), a configuration is used that separates the inputs and outputs onto two separate planes. This configuration features a two dimensional,

n-by-n Cartesian grid of inputs and a corresponding n-by-n grid of outputs. There are no hidden nodes and recurrence is disabled, so each of the n^2 input nodes has a link of a given weight to each of the n^2 output nodes (although weights can be zero, functionally eliminating the link). The parameter configurations for HyperNEAT and PNEAT are the same as in Gauci and Stanley (2007), except that the probability of adding a link was 30% (up from 10%) and the MutationPower, which controls the magnitude of weight mutations, was lowered from 2.5 to 0.1 after preliminary experiments revealed that this improved performance in the problem domains used in this paper. Every experimental trial within a treatment differed only in its random number generator seed, which influenced stochastic events such as mutations and the creation of randomized targets (either maps or weights). HyperNEAT and P-NEAT trials with the same random number seed in each treatment had the same targets.

3 Problem Domains and Results

The first problem, which will be called ‘Bit Mirroring,’ is intuitively simple 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 $\langle x_1=3, y_1=3 \rangle$ could be paired with output $\langle x_2=7, y_2=7 \rangle$). 1s and -1s are 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 a one, and values less than or equal to zero are considered a negative one. To reduce the effect of the randomness of the inputs, every generation each organism is evaluated on ten different sets of random inputs and these scores are summed to produce a fitness score for that organism. The max fitness is thus n^2 (one potential right answer for each input) $\times 10$.

The correct wiring is to create a positive valued link between each input node and its target output and, importantly, to zero out all links between each input node and non-target output nodes. That there is a correct wiring motif that needs to be repeated for each input cell creates an ‘inherent regularity’ to the problem. However, this inherent regularity is constant for a given grid size. The Bit Mirroring problem is challenging for evolutionary algorithms because links between input nodes and non-target nodes are likely to exist in initial random configurations, crop up through mutation, and can complicate fitness landscapes. Imagine, for example, that a mutation switches the weight value 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. While the problem is intuitively simple, it is not trivial.

Recall that highly regular problems are those where one motif must be repeated multiple times. One simple way to construct a highly regular Bit Mirroring problem is for the x and y values of the input and output nodes to be the same (i.e. the target is directly across). For example, $\langle x_1=5, y_1=6 \rangle$ should connect to $\langle x_2=5, y_2=6 \rangle$. There are at least three types of regularity in this highly regular version of the Bit Mirroring problem: 1) inputs and output targets have the same x values (they are in the same column), 2) inputs and output targets have the same y values (they are in the same row), and 3) the inherent regularity in the Bit Mirroring problem (see above). Each type of regularity can be scaled from high to low.

The first experiment uses a 7x7 grid and decreases the ‘within-column’ regularity by reducing the percentage of inputs whose target is constrained to be in the same column. Unconstrained nodes must have the same y value, but can have a different x value. 10 trials were performed for each treatment lasting 2000 generations. Fig. 1a reveals that the performance of HyperNEAT falls off as within-column regularity decreases. HyperNEAT is able to perfectly solve the problem in all but two trials of the most regular treatment, when the targets are all constrained to be directly across. As the within-column regularity decreases, the performance of HyperNEAT falls off fast. Interestingly, HyperNEAT does not benefit from the within-column regularity when 50% or fewer of its nodes are regularized in this way (only treatments with 60% or more column-constrained targets have fitnesses significantly better at a $p < .05$ level than fitness values from the treatment with 0% of nodes column-constrained; this and all future p values use Matlab’s Mann-Whitney U-test).

The second experiment, which also involved 10 trials lasting 2000 generations and a 7x7 grid, scales a similar but different type of regularity by allowing all targets to be random with respect to column, but decreasing the percent that are constrained to be in the same row (Fig. 1b). In a sense, experiment two picks up where experiment one left off. In fact, the least regular treatment from experiment one and the most regular treatment from experiment two have identical constraints (although different randomly generated mappings), which is why their distributions are similar. The performance of HyperNEAT also decreases as this type of regularity is diminished. Surprisingly, the pattern of degradation is similar to experiment one; HyperNEAT no longer provides a fitness boost due to within-row regularity once that regularity falls below 60% (p only $< .05$ comparing 0% row-constrained to $\geq 60%$ row-constrained treatments). While it is possible that running experiment one and two longer would have allowed significant differences to develop between less-regular treatments, it is relevant that no significant difference was present after 2000 generations, which is a substantial number in the field of evolving neural nets. It is interesting that the *range* of fitness values is also correlated with the regularity of the problem for HyperNEAT. This might be because, when regularity is present, the generative representation either discovered and exploited it, which would result in high fitness values, or it failed to fully discover the regularity, at which point its fitness more closely resembles less regular treatments.

Experiments one and two were also performed using P-NEAT, the direct representation control for HyperNEAT. As expected, the fitnesses produced by P-NEAT were not affected by the regularity of the problem. While for space constraints we only show P-NEAT values from experiment two (Fig. 1c), none of the P-NEAT treatments from experiment one were significantly different from P-NEAT treatments from experiment two ($p < .05$). Furthermore, within both experiments, none of the treatments were significantly different than that experiment’s 0% constrained treatment ($p > .05$). All HyperNEAT treatments from experiment one do significantly better than P-NEAT treatments from experiment one, due to both the within-row regularity present throughout and the inherent regularity of the Bit Mirroring problem. In experiment two, the within-row regularity decreases, leaving only the inherent regularity. However, presumably due to the inherent regularity of the problem, HyperNEAT still significantly outperforms P-NEAT on all treatment conditions except for the 20% constrained treatment. Computational constraints prevented the performance of more

trials, which may have eliminated this overlap in performance on the 20% constrained trial. While it is possible that running the trials for more generations would have allowed P-NEAT to catch up to HyperNEAT on irregular treatments in experiment two, viewing the fitness values across generations (not shown) suggests that, were this possible, the increase would have to be dramatic.

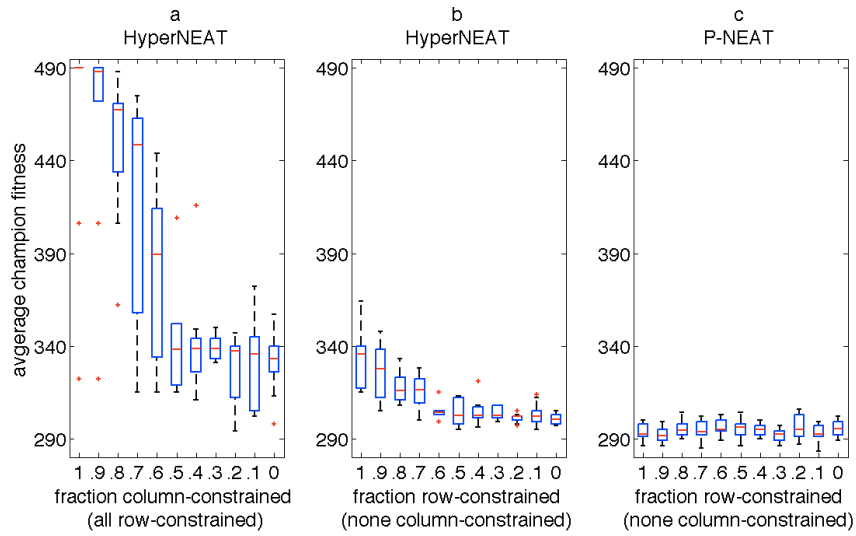


Fig. 1. HyperNEAT and P-NEAT on the Bit Mirroring problem as regularity decreases. **a)** HyperNEAT’s performance in experiment one, where within-column constraints are relaxed but within-row constraints remain. **b)** HyperNEAT’s performance in experiment two, where within-column constraints are eliminated and within-row constraints are relaxed. **c)** P-NEAT’s performance in experiment two, which is statistically indistinguishable from its performance on experiment one.

A third experiment continues the comparison of Hyper-NEAT to P-NEAT on problems of decreasing regularity. For this experiment trials lasted 2000 generations, as before, but we conducted 40 trials per treatment due to the high variance between trials. All targets in experiment three are random with respect to row and column, leaving only the regularity inherent in the Bit Mirroring problem. Since this inherent regularity stems from a motif that needs to be repeated for each input node (‘zero out links to all outputs but one’), the number of times this motif needs to be repeated decreases with the grid size. Unfortunately, there is no way to decrease this inherent regularity without also decreasing the problem complexity (i.e. the number of weights the algorithm is optimizing). Based solely on problem regularity, P-NEAT should perform better in comparison to HyperNEAT as this type of regularity is decreased. Fig 2. reveals that this is the case. The performance of HyperNEAT degrades to and then falls below that of P-NEAT as the grid size decreases. The overall decline is significant ($p < .05$ comparing the ratios on the 3x3 treatments vs. those 6x6 and greater). It is not clear why the trend is reversed on the smallest grid size. Note that

experiments one and two occurred on a 7x7 grid where the level of inherent regularity provided HyperNEAT an advantage over P-NEAT, even without within-column or within-row regularity, which explains HyperNEAT's superiority for those treatments reported above.

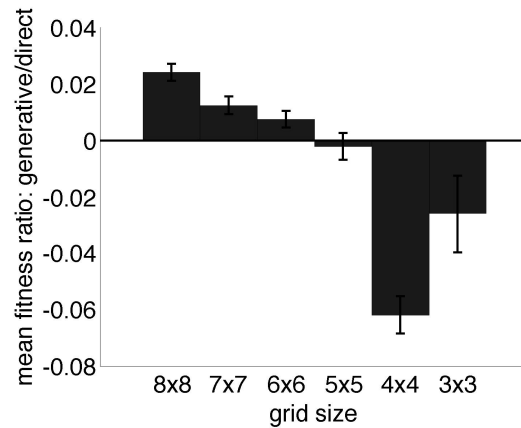


Fig. 2. Comparison of HyperNEAT to P-NEAT as the inherent regularity of the Bit Mirroring problem is decreased, which is accomplished by reducing grid size. Error bars show one standard error of the mean. Ratios are used instead of absolute differences because the allowable fitness ranges change with grid size.

Experiment three shows that once problems are sufficiently irregular and simple, the direct encoding P-NEAT can outperform the generative encoding HyperNEAT. The likely explanation is that HyperNEAT is biased towards creating modular phenotypes and has trouble when the problem features mostly exceptions and little rule. However, even the 3x3 version of the Bit Mirroring problem has some inherent regularity left over. This paper next compares HyperNEAT to P-NEAT on a problem that can be scaled to complete irregularity and where problem complexity remains constant. While there may always be regularities of which an experimenter is not aware, it seems that a completely irregular problem can be created if each link in the neural network phenotype has its own randomly chosen target value. In this 'Target Weights' problem, fitness measures how well the phenotype matches a pre-selected phenotype instead of evaluating a phenotype on a problem with inputs and outputs. A regular version of this problem can be constructed if all target weights are the same randomly chosen value. The regularity can be decreased by lowering the percent of weights that have a repeated target. For the treatment where 50% of the weights are repeated, for example, a 'repeated value' is randomly chosen and that value becomes the target weight for a randomly selected 50% of links. The remaining 50% of links each have a random target chosen independently. This experiment ran faster, allowing 10 trials per treatment of 5000 generations on a 3x3 grid.

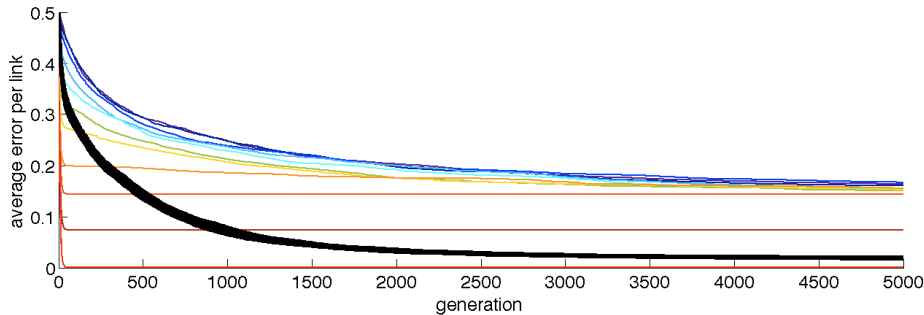


Fig. 3. Comparison of HyperNEAT to P-NEAT on 11 treatments of the Target Weights problem where the percent of targets repeated decreased by 10% from 100 to 0 percent. The initial average error of HyperNEAT (thin lines) is inversely related to the regularity of the treatment (e.g. the lowest line, with almost zero error throughout, is the most regular HyperNEAT treatment, and the highest line is the most irregular HyperNEAT treatment), although HyperNEAT's performance on less regular treatments becomes conflated over time. The performance of P-NEAT (thick lines) was not affected by the regularity of the problem, which is why the lines are overlaid and indistinguishable.

Fig 3 shows that HyperNEAT exploits the regularity of the Target Weights problem early on, but P-NEAT closes the gap fast and eventually outperforms HyperNEAT on all but the most regular treatment. This experiment provides another kind of example where a direct encoding does better compared to a generative encoding as the problem regularity decreases. This experiment also serves as a further illustration both that the performance of HyperNEAT decreases with the regularity of the problem, and that the fitness values, at least at the end of the run, are statistically indistinguishable once the regularity of the problem falls below a relatively high threshold ($p > .05$ comparing the 0% repeat treatment to all but the 90 and 100 percent repeat treatments). However, the difference in HyperNEAT's performance between treatments early on complicates the story of how HyperNEAT's performance flatlines below a certain regularity threshold, by making it depend on time. Fitness plots across generations from experiments one and two (not shown) do not tell a similar story; in those experiments the less regular treatments have similar fitness scores throughout. A further point of interest is the lack of progress HyperNEAT makes on the highly regular treatments (e.g. where 80 or 90 percent of the targets are repeated). While it exploits the regularity early on, HyperNEAT seems unable to make exceptions to the rule in order to encode the non-conforming link values, as evidenced by the lack of fitness improvements after the initial surge. Unsurprisingly, the P-NEAT trials from this experiment are statistically indistinguishable ($p > .05$).

Each of the previous experiments have shown how HyperNEAT and P-NEAT perform as a *single type* of regularity is scaled from high to low. Fig. 4 shows how HyperNEAT and P-NEAT perform as the number of *concurrent types* of regularity is decreased. It samples from the first four experiments. While it could have been the case that exploiting one type of regularity prevented the exploitation of others, Fig. 4

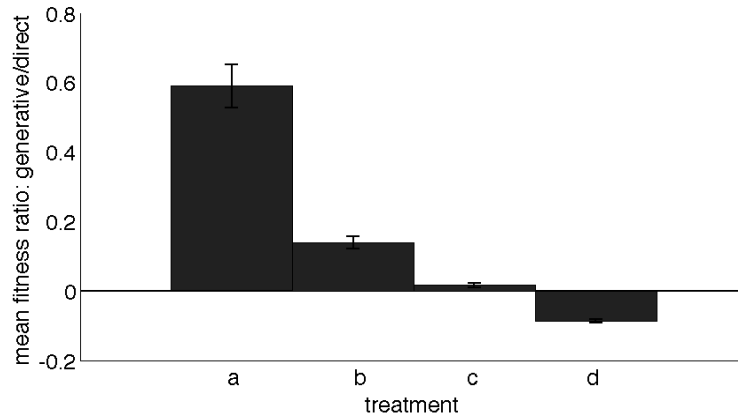


Fig. 4. Comparison of HyperNEAT to P-NEAT across experiments as regularity is decreased. **a)** All targets are constrained to be within the same column and row as their source on the 7x7 Bit Mirroring problem (three types of concurrent regularity). **b)** Targets are only constrained to be in the same row on the 7x7 Bit Mirroring problem (two types of concurrent regularity) **c)** Targets are randomly chosen, leaving only the inherent regularity of the 7x7 Bit Mirroring problem (one type of regularity) **d)** Randomly chosen (no repeated) values on the Target Weights problem (no types of regularity).

reveals that it is possible for HyperNEAT to simultaneously exploit multiple types of regularity. It also demonstrates that the performance of HyperNEAT degrades to, then falls below, that of P-NEAT as concurrent problem-regularity decreases.

4 Discussion, Future Work and Conclusion

The experiments in this paper, which cover four types of regularity from two different problems, paint a consistent picture despite some idiosyncrasies. In general, the HyperNEAT generative encoding showed some difficulty in making exceptions to the rules it discovered. Its performance decreased as problem regularity decreased. Nevertheless, the generative encoding did provide a fitness boost over its direct encoding counterpart on regular problems. The generative encoding's ability to simultaneously exploit concurrent types of regularities meant that the more types of regularity, the larger the boost. However, the generative encoding could only exploit a type of regularity when the amount of regularity within that type was relatively high. This result is not obvious from theoretical considerations and, to the authors' knowledge, has not been reported before. Future work is needed to see if the conclusions drawn from this generative encoding on these two problems apply to most generative encodings on many problems. It would also be interesting to test less extreme types of irregularity. Instead of non-constrained nodes being randomized, for example, they could be offset by a fixed amount. This would still test whether exceptions to the rule can be made, but would test a different, more regular, type of exception. Often the exceptions that need to be made to a rule do not involve radical departures from that rule, and generative encodings may do better at accommodating more subtle variations than those

tested here. Finally, while the direct encoding outperformed the generative encoding on irregular problems, as might be expected from the No Free Lunch theorem [12], it was not until the problem was relatively irregular that this transition occurred. P-NEAT only excelled on very small versions of the Bit Mirroring problem and the Target weights problem, where concurrent regularities were few. In fact, it was challenging for the authors to come up with problems irregular enough to provide an advantage to the direct encoding. Even the 7x7 Bit Mirroring problem, which is simple compared to real-world problems, had multiple regularities that could be exploited. It is likely that on most difficult real-world problems, the existence of many types of regularities will provide an advantage to generative encodings. One interesting question this paper raises, however, is whether the level of regularity within each type will be sufficient for a generative encoding to be able to exploit it.

References

1. Hornby, G.S., Pollack, J.B.: Creating High-Level Components with a Generative Representation for Body-Brain Evolution. *Artificial Life* 8(3), 223–246 (2002)
2. D’Ambrosio, D.B., Stanley, K.O.: A novel generative encoding for exploiting neural network sensor and output geometry. In: Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., Beyer, H.-G. (eds.) *GECCO 2007*, pp. 974–981. ACM Press, New York (2007)
3. Gauci, J.J., Stanley, K.O.: Generating Large-Scale Neural Networks Through Discovering Geometric Regularities. In: Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., Beyer, H.-G. (eds.) *GECCO 2007*, pp. 997–1004. ACM Press, New York (2007)
4. Gruau, F.: Genetic Synthesis of Boolean Neural Networks with a Cell Rewriting Developmental Process. *International Workshop on Combinations of Genetic Algorithms and Neural Networks* 6, 55–74 (1992)
5. Gruau, F., Whitley, D., Pyeatt, L.: A Comparison Between Cellular Encoding and Direct Encoding for Genetic Neural Networks. In: *Proc. 1st Ann. Conf. on Genetic Programming 1996*, pp. 81–89. MIT Press, Cambridge (1996)
6. Stanley, K.O., Miikkulainen, R.: A Taxonomy for Artificial Embryogeny. *Artificial Life* 9(2), 93–130 (2003)
7. Reisinger, J., Miikkulainen, R.: Acquiring Evolvability Through Adaptive Representations. In: Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., Beyer, H.-G. (eds.) *GECCO 2007*, pp. 1045–1052. ACM Press, New York (2007)
8. Nolfi, S., Miglino, O., Parisi, D.: Phenotypic Plasticity in Evolving Neural Networks. In: *Proc. Intl. Conf. from Perception to Action*. IEEE Press, Los Alamitos (1994)
9. Stanley, K.O.: Compositional Pattern Producing Networks: A Novel Abstraction of Development. *Genetic Programming and Evolvable Machines Special Issue on Developmental Systems* 8(2), 131–162 (2007)
10. Southan, C.: Has the Yo-Yo Stopped? An Assessment of Human Protein-Coding Gene Number. *Proteomics* 4(6), 1712–1726 (2004)
11. Stanley, K.O., Miikkulainen, R.: Evolving Neural Networks Through Augmenting Topologies. *Evolutionary Computation* 10(2), 99–127 (2002)
12. Wolpert, D.H., Macready, W.G.: No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation* 1, 67–82 (1997)