CPPNs Effectively Encode Fracture: A Response to Critical Factors in the Performance of HyperNEAT

Kenneth O. Stanley[1], (kstanley@cs.ucf.edu)
Jeff Clune[2] (jeffclune@uwyo.edu),
David B. D'Ambrosio[1] (ddambro@gmail.com),
Colin D. Green[3] (colin.green1@gmail.com),
Joel Lehman[4] (joel@cs.utexas.edu),
Gregory Morse[1] (gmorse@eecs.ucf.edu),
Justin K. Pugh[1] (jpugh@eecs.ucf.edu),
Sebastian Risi[5] (sebastian.risi@cornell.edu), and
Paul Szerlip[1] (pszerlip@eecs.ucf.edu)
[1]Department of Electrical Engineering and Computer Science
University of Central Florida, Orlando, FL 32816
[2]Department of Computer Science
University of Wyoming, Laramie, WY 82071
[3]Independent Researcher
[4]Department of Computer Science
The University of Texas at Austin, Austin, TX 78712
[5]Creative Machines Lab
Cornell University, Ithaca, NY 14853

University of Central Florida Dept. of EECS Technical Report CS-TR-13-05

Abstract

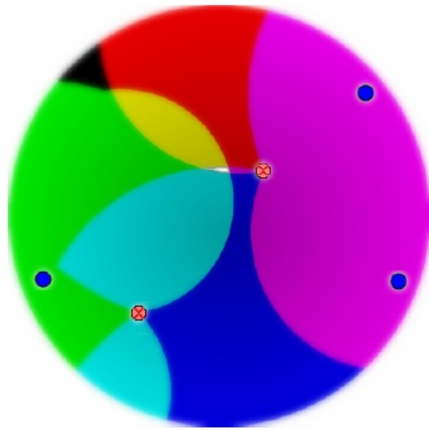
This paper demonstrates that compositional pattern producing networks (CPPNs) can produce phenotypic patterns that exhibit *fracture* (i.e. discontinuities in patterns) and that when neural networks are evolved with CPPNs, as in the Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) approach, the algorithm effectively incorporates hidden nodes to improve performance. These findings contradict those of a recent paper by van den Berg and Whiteson [51], to which this paper is in part a response. In particular, while van den Berg and Whiteson [51] report difficulty for HyperNEAT in several experimental domains, this paper provides counter-evidence suggesting that HyperNEAT in fact performs well in these domains. It also examines the reasons for the discrepancy, which is largely a result of implementation details. Building on this foundation of counter-evidence in the domains from van den Berg and Whiteson [51], the paper then shifts to refuting their central hypothesis, which is that patterns with fracture are problematic for CPPNs. This hypothesis is contradicted through the presentation of a wide variety of detailed examples of fracture

in CPPNs, suggesting that fracture in CPPNs may be the norm rather than the exception. In this way, this paper goes beyond examining particular experimental domains by highlighting the largely unexplored opportunity to study fracture in CPPN-based patterns, which can potentially initiate exciting future research into representation.

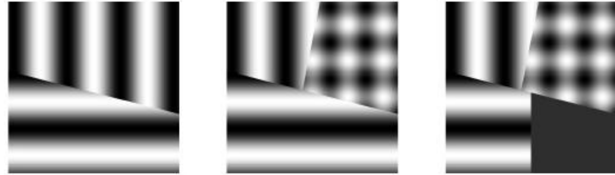
1 Introduction

Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) [22, 49, 52] is a recent method for evolving large-scale, regular artificial neural networks that has been applied in a wide variety of domains [2, 3, 5, 8, 9, 10, 11, 13, 14, 15, 17, 18, 19, 21, 22, 25, 29, 52]. Some highlights include multiagent robot control [17], checkers [21, 22], quadruped locomotion [15, 41, 43], Robocup Keepaway [52], and simulated driving in traffic [19]. A recent paper by van den Berg and Whiteson [51] suggests that HyperNEAT struggles in domains in which it would be expected to perform well, raising an important question about its capabilities. In direct contradiction to these results, reproductions in this paper of two of the three experiments in van den Berg and Whiteson [51] (with source code provided at http://eplex.cs.ucf.edu/cfresponse13.html) demonstrate that HyperNEAT in fact *can* succeed effectively in the domains in question, suggesting that the conclusions drawn from the original experiments may be misleading. The third domain, the walking gait task, is not reimplemented but positive results in a similar domain are reviewed. This paper also provides evidence for the ability of compositional pattern producing networks (CPPNs) (which are the encoding in HyperNEAT) to encode a property called *fracture*, which van den Berg and Whiteson [51] hypothesize poses a problem for CPPNs.

In particular, the hypothesis advanced by van den Berg and Whiteson [51] to explain the apparent underperformance by HyperNEAT in three key domains is that "fracture can be highly problematic for HyperNEAT." Thus the concept of fracture, which van den Berg and Whiteson [51] define as "discontinuous variation of patterns," is central to the main hypothesis. Fracture was originally defined by Kohl and Miikkulainen [30] as a "highly discontinuous mapping between states and optimal actions" within a neural network, but van den Berg and Whiteson [51] generalize its definition to make it applicable to HyperNEAT because HyperNEAT evolves a CPPN [48] that indirectly maps to a final neural network controller. That is, rather than producing a mapping between states and actions, in HyperNEAT the CPPN outputs a connectivity pattern, which is why "discontinuous variation of patterns" captures the idea of fracture better for HyperNEAT. Both Kohl and Miikkulainen [30] and van den Berg and Whiteson [51] provide visualizations of fractured patterns, which are reproduced here to help to illustrate the idea (figure 1).



(a) A fractured pattern [30]



(b) Increasingly fractured patterns [51]

Figure 1: **Illustrations of Fracture.** These images, reproduced from Kohl and Miikkulainen [30] and van den Berg and Whiteson [51], give an intuitive view of fractured patterns. The blue dots and red dot in (a) represent potential agent positions in a keepaway soccer control problem in which the right decision depends on which fractured regions contain the agents. The sequence in (b) suggests increasing fracture accumulating over generations.

The main approach of van den Berg and Whiteson [51] is first to show that HyperNEAT fails to perform effectively on three tasks that are slightly harder variants of easier tasks (which do not require evolving a complex CPPN) on which HyperNEAT previously succeeded. These failures are then attributed to the hypothesized problem with evolving fracture. To explain why fracture in particular might account for these failures, van den Berg and Whiteson [51] connect the failures to the idea that fractured problems might require more than a trivial number of nodes: "The reason we hypothesize that fracture is difficult for HyperNEAT is that it seems that many nodes are needed to divide the substrate into such regions." Thus the underlying implication is that it may be difficult to evolve patterns with CPPNs that require many nodes to represent, and that the domains in which HyperNEAT appears to falter may require evolving such patterns.

Contrary to these claims, this paper begins by showing first that HyperNEAT actually does not fail in two reimplemented domains introduced by van den Berg and Whiteson [51], thereby suggesting a more nuanced story. The difference in results is likely attributable to implementation details, which are pointed out in this paper, but the more important point is that there is no fundamental problem for HyperNEAT in these domains. After demonstrating HyperNEAT succeeding in these domains, the focus will then turn to showing a variety of evidence for effective evolution of fracture through CPPNs, thereby raising further doubt on the hypothesized problem with fracture. The paper then concludes with a discussion of the interpretation of negative results in the field of generative and developmental systems.

2 Experimental Reproductions

This section reexamines the three domains that van den Berg and Whiteson [51] claim to show are difficult for HyperNEAT: (1) visual discrimination with triangles, (2) line following with vertical gray bands, and (3) top-heavy quadruped locomotion. For the first two domains, the experiments are reproduced in their entirety. For the quadruped domain, other published results from HyperNEAT with similar quadrupeds are reviewed.

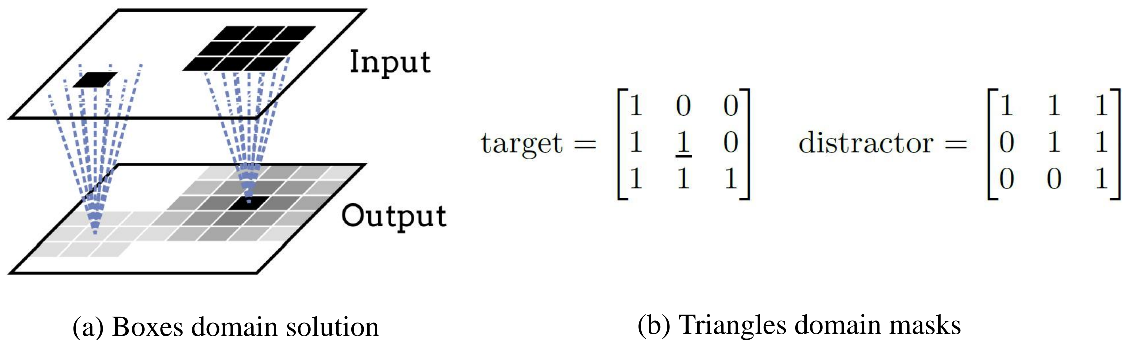
The main idea behind these three domains is that each is slightly harder than a variant that HyperNEAT can solve with no hidden nodes in the CPPN. Thus when van den Berg and Whiteson [51] show HyperNEAT failing in these harder variants, it implies that when hidden nodes become necessary, HyperNEAT begins to exhibit problems. In contrast, this section will show that HyperNEAT has no fundamental difficulties with these harder domains.

2.1 Visual Discrimination with Triangles

This task is a variation of the original "boxes" visual discrimination task that was used to demonstrate the scalability of HyperNEAT to very large substrate resolutions [49]. The substrate in the boxes version includes an 11×11 input layer that is a two-dimensional visual field and an output layer of the same size. Two boxes (size 3×3 and 1×1) are placed on the input layer at arbitrary locations. The objective of the domain is that the output layer should activate the node at the same (x, y) location as the center of the bigger box with highest activation (figure 2a). In this way, the domain tests the ability of the evolved substrate to *discriminate* between the big and the small box.

The main difference in the "triangles" variant from van den Berg and Whiteson [51] is that instead of discriminating a big box from a small box, the substrate must discriminate the location of a target triangle with a certain orientation from the location of a distractor triangle of the same size but with a different orientation (figure 2b). The triangle pairs must be discriminated by each candidate neural network at 75 different relative locations that are determined randomly during each training evaluation. This variant of the task is harder than the boxes variant because the substrate cannot simply correlate activation level to the number of active pixels nearby (which would work in boxes). Instead it must learn to focus only on orientation. Solution quality is measured in this task as the *distance* between the point of highest activation and the center of the target triangle.

The triangles variant is interesting because van den Berg and Whiteson [51] hypothesize that this in-



(a) Boxes domain solution

(b) Triangles domain masks

Figure 2: **Boxes and Triangles Domains.** Depictions of the boxes domain (a) and the masks for the triangles variant (b) are shown (reproduced from van den Berg and Whiteson [51]).

creased difficulty requires hidden nodes, an idea that their results support. They show this by comparing normal HyperNEAT, in which the number of hidden nodes increases over evolutionary time, to a version of HyperNEAT restricted to have no CPPN hidden nodes. HyperNEAT is shown to perform only marginally better when the number of hidden nodes is not restricted, suggesting that allowing it to evolve more hidden nodes provides no advantage. The experiments by van den Berg and Whiteson [51] with HyperNEAT in the triangles domain are run with their own implementation of HyperNEAT called *PEAS-NEAT*.

Additionally, van den Berg and Whiteson [51] introduce a simple *Wavelet Encoding* to provide a performance baseline to compare to HyperNEAT. This Wavelet approach sums a number of wavelet basis functions with genetically-specified parameters to determine the connectivity pattern (as opposed to specifying connectivity via a CPPN as in HyperNEAT). In effect, the Wavelet Encoding is similar to a direct encoding because it places weighted independent oscillating patterns that are confined to a limited area at explicitly enumerated locations within a pattern; in other words, it cannot compose patterns hierarchically to create higher-level patterns the way CPPNs do. However, it shares some properties with indirect encoding because the area of each wavelet can encompass multiple connections in the substrate and thus correlations between connections may result. A similar method of encoding networks through the frequency domain (using a discrete cosine transform) was previously introduced by Gomez et al. [23]. In any case, the main purpose of the Wavelet Encoding is not to promote a new encoding but rather to make the point that because it is reported to perform better than HyperNEAT in the triangles domain, HyperNEAT should have been able to perform similarly well if it could make effective use of hidden nodes. Specifically, van den Berg and Whiteson [51] report that the PEAS-NEAT version of HyperNEAT on average achieves a distance of about 2.5, while Wavelets perform better at about 1.3 on average.

Thus the main questions raised by this experiment are (1) whether HyperNEAT cannot perform well in this domain and (2) whether HyperNEAT cannot make effective use of hidden nodes in this domain.

To examine these questions, two new experiments are implemented here. The first experiment aims to show that it is difficult to conclude broad incapacities for a method (or even a particular implementation) from only a single setup. This first experiment thus simply re-runs the PEAS-NEAT setup with higher structure-adding mutation rates. In particular, the triangles task is attempted again with PEAS-NEAT with three different structural mutation rates; 10 runs are completed at each rate. The *low* rate is the original rate used by van den Berg and Whiteson [51] (0.1 probability for adding a connection and 0.03 of adding a node), which yields poor results. The *medium* rate is instead 0.15 and 0.06, and the *high rate* is 0.3 and 0.1. The theory behind trying higher rates is that if the hypothesis is that HyperNEAT has trouble adding structure, then perhaps the problem is simply that too little structure is being added (as opposed to any deeper problem with the algorithm). Because PEAS-NEAT may be updated occasionally by its authors at https://github.com/noio/peas, the exact implementation used for this experiment is included at http://eplex.cs.ucf.edu/cfresponse13.html so that it can be reproduced.

The main result is that indeed increasing the mutation rate boosts the average performance of PEAS-NEAT to about 1.75, which is indeed a better performance than the 2.5 reported for HyperNEAT by van den Berg and Whiteson [51] (figure 3). Thus much of the reported performance difficulty may simply be a result of mutation rates that are too low. This result suggests a useful lesson: If HyperNEAT appears to be experiencing difficulty adding structure, simply increasing the structural mutation rate may help to rectify the problem.

Nevertheless, these results do not *prove* that HyperNEAT makes effective use of hidden nodes in this task because it is still possible that hidden nodes are not actually needed. Also, it would be interesting if performance can be raised even to a point better than the Wavelet implementation. To investigate these issues further, the triangles domain was reimplemented with the SharpNEAT 2.0-based version of HyperNEAT [24]. Reimplementing the task in this other version of HyperNEAT makes it possible to examine HyperNEAT's performance in a well-tested and widely-used implementation that is popular with many users of NEAT and HyperNEAT.[1] The full source code for the reimplementation is available at http://eplex.cs.ucf.edu/cfresponse13.html. The only known difference between these implementations is that in the SharpNEAT-based implementation, triangles are never placed directly adjacent when they are placed at random locations during training (which conforms to the original convention in the boxes domain), although in PEAS-NEAT they sometimes are. The motivation for avoiding such touching is that two triangles touching each other yields an ambiguous configuration (because it is no longer shaped like a trian-

[1] Colin Green maintains a list of published research based on SharpNEAT at http://sharpneat.sourceforge.net/.

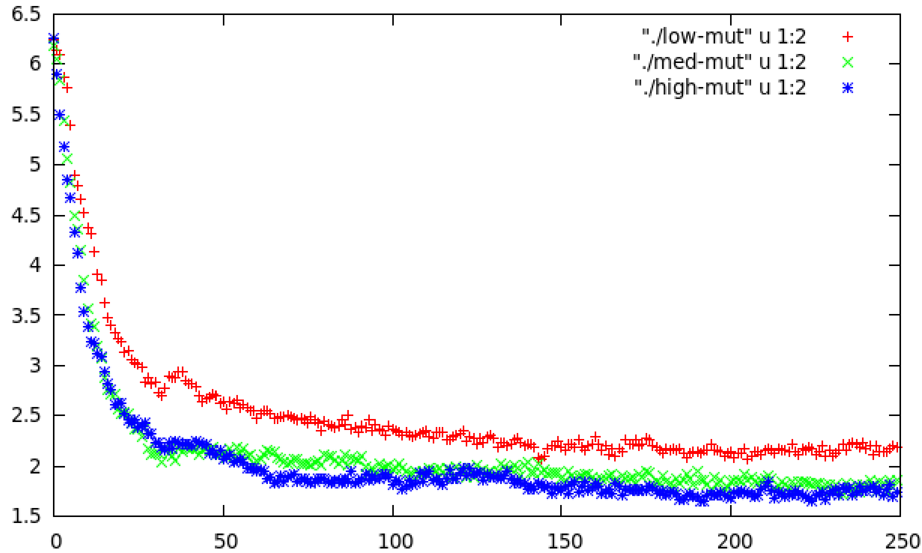
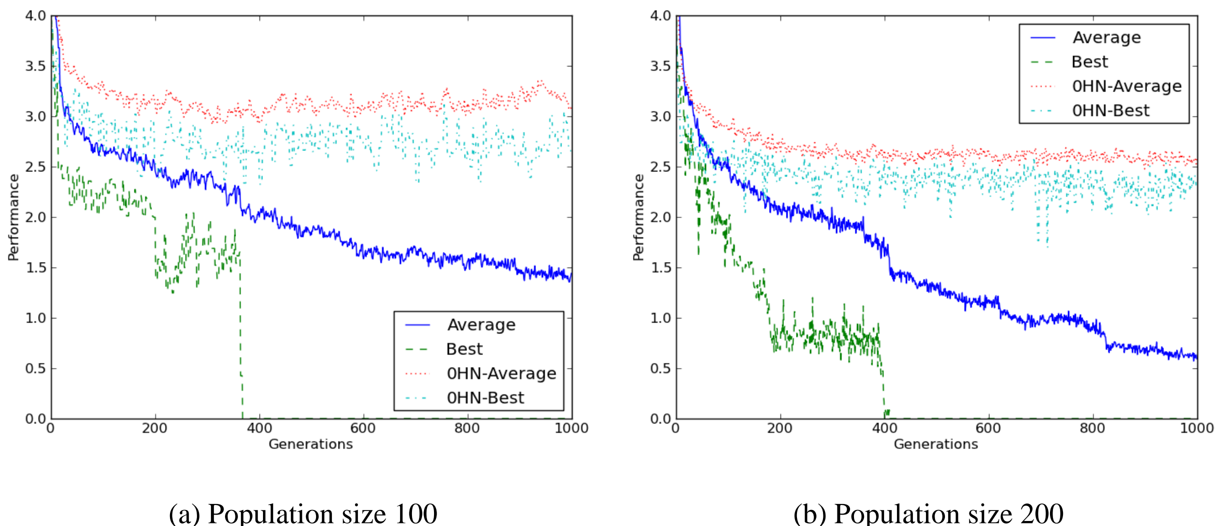


Figure 3: **Performance on Triangles in PEAS-NEAT with Different Structural Mutation Rates (lower is better).** The *low* (upper) line reproduces the poor performance of PEAS-NEAT reported with its original low structural mutation rate. However, the two lower lines (which are close), representing *medium* and *high* structural mutation rates, both exhibit good performance.

gle) that can therefore in effect reduce training performance. This experiment also uses structural mutation rates that worked well in previous SharpNEAT-based experiments: 0.05 probability for adding both nodes and connections (full parameters are given at http://eplex.cs.ucf.edu/cfresponse13.html).

Other considerations that can potentially alter results are the length of the run and the size of the population (the larger the population, the more HyperNEAT can take advantage of the speciation mechanism in NEAT to explore diverse CPPN topologies). That is, results might be different if the experiment is run for more generations or with a larger population. For example, the danger with the hypothesis that HyperNEAT has trouble effectively incorporating CPPN hidden nodes is that longer runs might exhibit the opposite result. Therefore, ten runs with the SharpNEAT-based implementation with populations of both 100 (which is the size used by van den Berg and Whiteson [51]) and 200 are given 1,000 generations instead of the 250 generations allowed by van den Berg and Whiteson [51]. That way, genuine long-term dynamics can be observed. After all, incorporating structure requires building up enough parts to solve the problem, which can take time even when the algorithm is working effectively.

As shown in figure 4, the main result is indeed that HyperNEAT without restrictions on the number of hidden nodes increasingly outcompetes the variant restricted to zero hidden nodes. At the end of 1,000 generations, the difference is highly significant ($p < 0.001$ from Student's t-test) for both population sizes. While the zero-hidden-node variant stagnates above 2.5 regardless of population size, with hidden nodes



(a) Population size 100

(b) Population size 200

Figure 4: **Performance Comparison on Triangles in SharpNEAT-based HyperNEAT (lower is better).** The *Average* and *Best* lines in both graphs (which appear at bottom because their performance is superior) give the performance of regular HyperNEAT (with no node restrictions) over generations. The *0HN-Average* and *0HN-Best* show the performance when HyperNEAT is restricted to zero hidden nodes. Both population sizes were run ten times. For both sizes, HyperNEAT without the node restriction ultimately performs significantly better than when restricted to zero hidden nodes ($p < 0.001$ at both sizes 100 and 200). In fact, the best networks from HyperNEAT without node restrictions sometimes exhibit *perfect* performance, while 0HN networks never come close to a perfect evaluation, demonstrating the critical importance of hidden nodes in this problem and HyperNEAT's success at utilizing them effectively.[3]

HyperNEAT achieves an average performance of 1.4 with the smaller population and 0.6 with the larger. (The final average Wavelet baseline reported by van den Berg and Whiteson [51] is about 1.3, which is about twice as inaccurate as the average HyperNEAT with the larger population. Of course, as with HyperNEAT, it is possible that Wavelets could improve given a larger population or more generations.) These results establish definitively in contrast to the hypothesis of van den Berg and Whiteson [51] that HyperNEAT *can* gain a significant edge by evolving CPPN hidden nodes.

Furthermore, interestingly, some runs of the unrestricted variant of HyperNEAT eventually evolve solutions that exhibit *perfect* performance across their 75 evaluation trials (in particular three runs of ten exhibit perfect performance at size 100 and five runs of ten at size 200). It is not reported in van den Berg and Whiteson [51] whether perfect performance is ever achieved during training by any approaches. The average number of hidden nodes in such perfectly-performing solutions is 20.6 ($stdev = 8.0$). Given that

[3]The performance of the 0HN variant in these experiments is slightly worse than in van den Berg and Whiteson [51], which may be because van den Berg and Whiteson [51] provided their CPPN output nodes access to activation functions other than sigmoids, while traditional HyperNEAT implementations restrict outputs only to sigmoids so that output is always guaranteed to be signed. However, of course, the important result in any case is the success of the unrestricted variant.

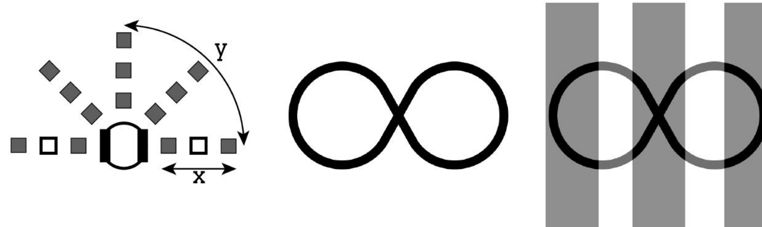


Figure 5: **Line Following Domain (reprinted from van den Berg and Whiteson [51]).** The sensor configuration (left) for the driver is shown next to the track from the easier version of the problem (middle) and the more challenging one (right) with gray distractor bands running vertically across the track.

none of the zero-hidden-node runs find such a solution, adding at least one hidden node appears *essential* in this domain to achieving perfect training performance. That HyperNEAT evolves such essential structure suggests that HyperNEAT can evolve hidden nodes when they are necessary.

Overall the results of this reexamination of the triangles domain show that (1) HyperNEAT can perform well in this domain, even sometimes performing perfectly during training, and that (2) HyperNEAT makes effective use of CPPN hidden nodes to increase its performance. Rather than suggesting a fundamental problem with evolving hidden nodes or fracture in HyperNEAT, these results suggest that higher structural mutation rates, longer runs, and increased population sizes all can help to yield success for HyperNEAT in this domain. Thus this domain actually turns out to be an effective platform for validating HyperNEAT's ability to evolve and take advantage of CPPN hidden nodes.

2.2 Line Following with Vertical Gray Bands

The line following domain variant is inspired by experiments originally by Buk et al. [8] and Drchal et al. [19] in which an agent must evolve to drive a simulated vehicle along a track while avoiding straying off the track, where driving is significantly slowed (figure 5 middle). The idea in the version with gray bands (figure 5 right) is that the agent sensors (figure 5 left) cannot rely on a constant shade of white in the area outside the track, making the problem harder.

The hypothesis in van den Berg and Whiteson [51] is that this lack of consistency in the background necessitates hidden nodes in the CPPN to solve the problem effectively. Their experimental results with PEAS-NEAT suggest accordingly that HyperNEAT indeed does not perform as well as the Wavelet baseline approach (explained in the previous section) and performs no better than a version of HyperNEAT restricted to only one hidden node. Fitness is measured (and reported) as average *speed*.

A potentially confounding factor with the speed statistic used by van den Berg and Whiteson [51]

(which comes from the original line following experiments [8, 19]) is that it does not explicitly reward covering the whole track.[4] For example, although the track is a figure-eight, it is possible in this domain to achieve a high speed simply by remaining within *one* of the two loops in figure 5. In fact, testing the PEAS-NEAT software indeed confirms that Wavelet solutions sometimes remain on only one side of the figure-eight (videos from the final generation PEAS-NEAT showing this behavior are available at http://eplex.cs.ucf.edu/cfresponse13.html), which can score highly with the speed statistic even thought it is not a behavior that would be expected to correlate to high fitness. Thus an interesting question not discussed in van den Berg and Whiteson [51] is whether HyperNEAT drivers are able to cover the whole figure-eight.

On the other hand, whether or not the drivers remain on only one side, the reported result is still that HyperNEAT evolves slower drivers. However, the problem identified to *explain* the apparent slowness, which is that HyperNEAT-evolved drivers often veer off the track, is the most important issue. As explained by van den Berg and Whiteson [51], "Examination of the resulting controllers shows that those generated by HyperNEAT frequently adjust their trajectory only when they are already off-road." Thus the key question is whether HyperNEAT can evolve drivers that stay within the lines consistently despite the distractor bands.

To test whether staying within the lines is indeed problematic for HyperNEAT in general (or whether the reported performance may be an artifact of the particular combination of PEAS-NEAT and its line-following domain implementation), the line-following domain was reimplemented to run with SharpNEAT [24]. Full source code for the reimplementation, including all parameter settings, is available at http://eplex.cs.ucf.edu/cfresponse13.html. Given the advantage seen with a larger (200) population in the triangles experiment, a population of 200 is also used for this experiment. Ten runs of HyperNEAT were attempted without any restriction on hidden nodes and ten were attempted with CPPNs restricted to zero hidden nodes. As in van den Berg and Whiteson [51], speed is measured as the total distance covered divided by the simulation time. However, it is important to note that speed values cannot be compared directly between experiments because the physics behind the simulators is likely slightly different. Nevertheless the problem of navigating through regions with different background shades remains the pivotal challenge in both experiments.

The main result is that HyperNEAT solves the problem of evolving a competent driver in the hard version of the task (i.e. with vertical distractor bands) with ease (figure 6), usually within 20 generations. In fact, drivers evolved by HyperNEAT shown in videos at http://eplex.cs.ucf.edu/cfresponse13.html appear

[4] It is important to note that this difference was pointed out to us in personal correspondence from the authors on May 27th, 2013.

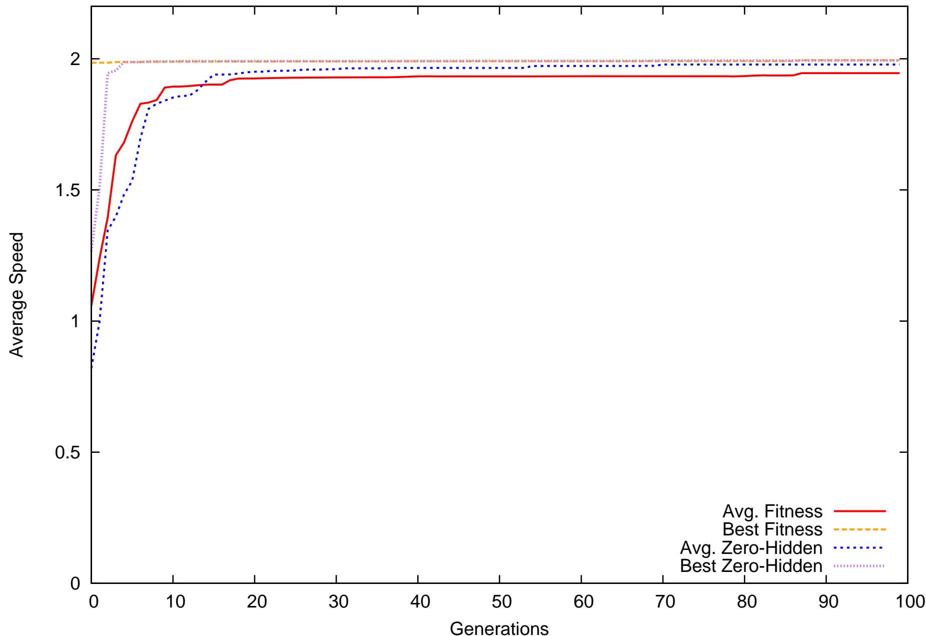


Figure 6: **Performance in Hard Line Following by SharpNEAT-based HyperNEAT.** SharpNEAT-based HyperNEAT achieves near-perfect performance in this domain on average whether or not it is restricted to zero hidden nodes. (*Avg. and Best* denote average and best performance in runs in which hidden nodes are not restricted while the number of hidden nodes is restricted to zero in *Zero-Hidden* runs.) Ten runs of each treatment are averaged. The task is usually solved within 20 generations.

smoother than typical Wavelet drivers from PEAS-NEAT (also shown in videos at the website), although of course the domains are implemented differently.

While a direct comparison of speed results with van den Berg and Whiteson [51] is not possible because of the potential simulator differences, the *qualitative* results are the important issue in this domain. In particular, it is clear that HyperNEAT-based drivers in the reimplementation do *not* “frequently adjust their trajectory only when they are already off-road” because they almost never go off-road. In fact, in nine of ten runs with hidden nodes (and nine of ten runs without hidden nodes) the champions *never* go off road, exhibiting perfect continuity across both white and gray patches (figure 7). The conclusion is that the PEAS-NEAT implementation of HyperNEAT may not be representative of the performance of other HyperNEAT implementations on this problem; in fact, in the SharpNEAT implementation the problem appears trivial for HyperNEAT to solve.

Also interestingly, recall that the speed-based fitness measure used by van den Berg and Whiteson [51] and reused here does not explicitly reward covering the whole track. In fact, Wavelet-based champions evolved in PEAS-NEAT confirm that speed-based fitness does not ensure the whole track is covered (see videos). However, in seven out of ten runs with hidden nodes and eight out of ten runs without hidden

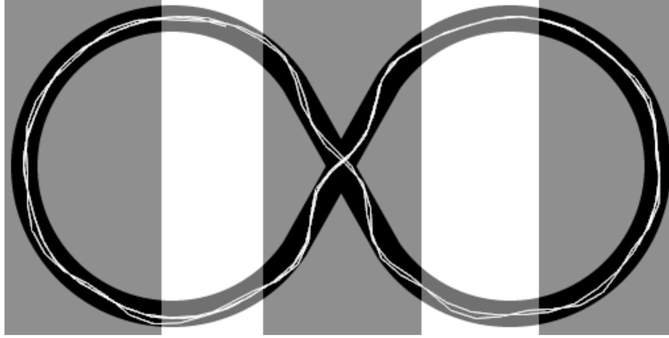


Figure 7: **Typical Path Followed by Evolved HyperNEAT Driver.** In the depicted path, the driver never strays outside the path. Such perfect navigation is exhibited by HyperNEAT in the reimplemented domain in 90% of runs. It also usually covers the entire figure-eight, which was achieved in 70% of runs with hidden nodes and 80% of runs without hidden nodes, even though coverage was not explicitly rewarded.

nodes in the SharpNEAT-based reimplementation, HyperNEAT-evolved drivers cover the whole figure-eight perfectly anyway. This result shows that evolving drivers that cover the track, stay within the lines, and avoid confusion from the distractors is not difficult for HyperNEAT.

However, these results also expose that a good solution is so easy to obtain in this problem that it does not require CPPN hidden nodes. That is, runs without hidden nodes perform as well as those that are allowed to evolve hidden nodes. Therefore, the fact that it ultimately does not require such nodes means it is not a suitable test for answering the question of whether HyperNEAT can evolve hidden nodes when needed. Nevertheless, at least the results show it is easy for HyperNEAT to evolve drivers that never go off the road and cover the full track. These successful drivers (as shown in the videos) also spend most of their time at maximal or near-maximal speed. Furthermore, the average number of hidden nodes evolved to solve the task perfectly when hidden nodes are allowed to evolve is only 1.75 ($stdev = 0.97$), suggesting that HyperNEAT does not add an excessive number of hidden nodes when they are not needed, as is claimed in van den Berg and Whiteson [51]. (Recall that the average number of hidden nodes for perfect solutions in the triangles problem, which *does* require hidden nodes to solve perfectly, is 20.6. It is not clear whether all of those nodes are necessary, but the difference between the number of nodes in the respective domains suggests that HyperNEAT does tailor the number of hidden CPPN nodes to the problem, and does not add nodes when they are unnecessary.)

It is not entirely clear why HyperNEAT in PEAS-NEAT appears worse on the line following task than SharpNEAT's HyperNEAT. There are many implementation differences between the two software packages that could account for the performance gap. It is also possible that PEAS-NEAT would perform better in the version of the domain used with SharpNEAT. Finally, it is possible that the larger population size or

different mutation rates could help PEAS-NEAT (as in the triangles domain).

So far reexaminations of triangles and line following have shown that HyperNEAT indeed solves triangles effectively by incorporating hidden nodes (contrary to the outcome in van den Berg and Whiteson [51]), and that it also solves the line following task without difficulty (also contrary to the originally reported results). Together these results suggest HyperNEAT is well-suited to these domains and furthermore that it can summon the appropriate hidden nodes for the task.

2.3 Top-heavy Quadruped Locomotion

The objective in the quadruped task (also called the *walking gait* task) is to evolve a controller for a quadruped robot that allows it to walk. The controller directs the motion of two hips joints and one knee joint in each of the four legs (12 degrees of freedom). The ability of HyperNEAT to succeed in this domain was first demonstrated by Clune et al. [15], but van den Berg and Whiteson [51] report that when they rerun the experiment with Clune's source code, the predominant solution synchronizes all joints to match each other, yielding a simple *hopping* behavior. In fact, the hopping behavior is so simple that the authors find it can be evolved easily with zero CPPN hidden nodes. The implication is that HyperNEAT does not evolve more sophisticated gaits because it cannot effectively optimize the multiple hidden nodes that would be needed to do so. It should be noted, though, that in Clune et al. [14] additional variants also evolved, including left-right and front-back symmetry.

To validate the hypothesis that HyperNEAT cannot evolve hidden nodes to produce complex gaits, van den Berg and Whiteson [51] modify the task by increasing the weight of the quadruped torso by a factor of ten, creating a top-heavy quadruped. The idea is that hopping with such a body should be significantly less effective and less stable, requiring a different kind of gait for good performance. Results with the modified robot seem to confirm this intuition, showing that HyperNEAT performs suboptimally in the new body because it cannot seem to find any non-hopping gaits. It also performs no better than a version of HyperNEAT restricted to zero hidden nodes, also adding weight to the hypothesis.

However, the revisited results in the previous two experiments in this paper hint that once again the apparent problem in this domain is potentially not as fundamental as a problem evolving CPPN hidden nodes. Rather, it is reasonable to hypothesize (given the results in the previous two domains) that the tendency towards hopping observed by van den Berg and Whiteson [51] is once again an artifact of the particular implementation or the parameter settings not being tuned for the specific problem. Lending significant support to this hypothesis are recent publications showing HyperNEAT consistently evolving nontrivial gaits

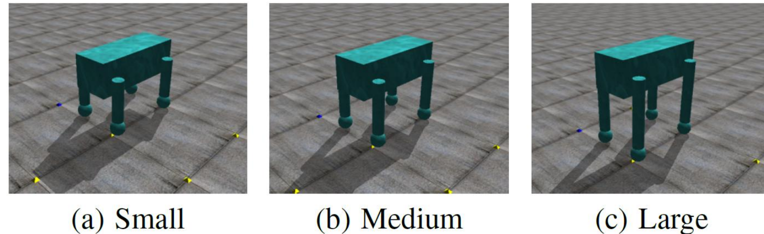


Figure 8: **Multiple Quadruped Morphologies (reproduced from Risi et al. [43]).** Not only does HyperNEAT evolve a single quadruped controller in Risi et al. [43], but it evolves a single CPPN that can generate controllers for a range of morphologies, including those shown here.

in similar top-heavy quadrupeds [41, 43]. In fact, Risi et al. [43] evolves quadruped gaits in a significantly more difficult context: Not only must the CPPN encode the substrate for a single quadruped morphology, but it must be able to generate working controllers for *multiple* morphologies (figure 8). Furthermore, it must generate weights for a more complicated continuous-time recurrent neural network (CTRNN) architecture, as opposed to the simpler sine-fed architecture of Clune et al. [15]. In other words, the controller in Risi et al. [43] must generate its own oscillations internally. Nevertheless, with all these obstacles, it consistently generates fluid and nontrivial gaits (i.e. with alternating leg motions) for quadrupeds of multiple sizes. Videos of these results can be seen at http://youtu.be/oLSSt5GyHNk.

Thus although time constraints prevented us from implementing this third domain, especially in light of the fact that such a reimplementation has been done in two similar, but arguably harder variants of the top-heavy quadruped problem, the problem reported by van den Berg and Whiteson [51] with the top-heavy quadruped may not be representative of a general issue with HyperNEAT (like with the first two domains). Of course, further experimentation is necessary to establish whether this hypothesis is correct.

3 Fracture in CPPNs

While van den Berg and Whiteson [51] report difficulty for HyperNEAT in three domains, the reexamined experiments in the preceding section show why it can be challenging to draw general inferences from negative results. Instead of a deep underlying pathology, a mutation rate may simply be too low, or runs may be terminated too early. Slight changes to the fitness function might help to reduce deception in the domain, or other aspects of experimental setup could inadvertently make a difference. Each of these possibilities is interesting in its own right. For example, sensitivity to parameters is a legitimate and important issue. However, drawing fundamental inferences about the method itself can be tricky as a result.

Nevertheless, van den Berg and Whiteson [51] are inspired by the weak performance they observe

to hypothesize a fundamental problem for HyperNEAT with encoding fracture (figure 1) when fracture is needed. Because CPPNs encode the patterns of connectivity in HyperNEAT, this hypothesis in effect entails the inability of CPPNs to evolve a representation of fractured patterns (which would also mean that HyperNEAT-encoded neural networks would struggle to exhibit behavioral fracture, i.e. performing different behaviors in different regions of the state space). The authors discuss this hypothesized problem with CPPNs at some length, concluding that it is probably because representing fracture requires many nodes:

> The reason we hypothesize that fracture is difficult for HyperNEAT is that it seems that many nodes are needed to divide the substrate into such regions.

In contrast to this hypothesis, given that the two reimplemented experiments in this paper show that HyperNEAT *can* solve the problems posed by van den Berg and Whiteson [51] without significant difficulty, it is possible therefore that encoding fracture is *not* problematic for CPPNs. For example, if the problems with performance are remedied with simple changes implementation details (as they now appear to be), then it is plausible that there is no fundamental problem after all.

In fact, separately from the experiments in van den Berg and Whiteson [51], a wide range of evidence for fracture in CPPN-encoded patterns already exists. Accordingly, this section surveys this evidence, which provides further counterevidence to the hypothesis that CPPNs have a problem with fracture.

3.1 Fractured Patterns in HyperNEAT Networks

Fracture can be visualized in neural networks encoded by CPPNs by displaying connection weights within the geometry of the network. For example, Coleman [16] evolves HyperNEAT networks that can identify the locations of various shapes in a visual field (a domain similar to the triangles domain in this paper). Visualizations of the connectivity of these networks from Coleman [16] (figure 9) exhibit explicit fracture: Each visual field (depicted as incoming weights to individual neurons in the hidden layer) exhibits two distinct weight gradients that fracture in the middle of the field. In fact, this fracture is reminiscent of the visualization provided by van den Berg and Whiteson [51] of patterns that are hypothesized to be problematic for CPPNs to encode.

The weight patterns of three-dimensional neural network substrates encoded by evolved CPPNs can also be visualized. Clune et al. [15] shows several such three-dimensional weight patterns evolved by HyperNEAT with evident fracture (figure 10).

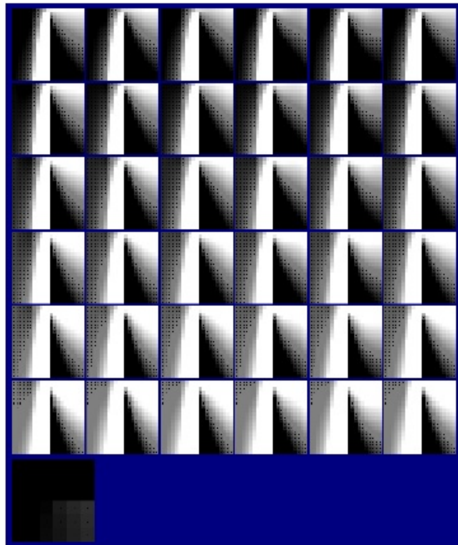


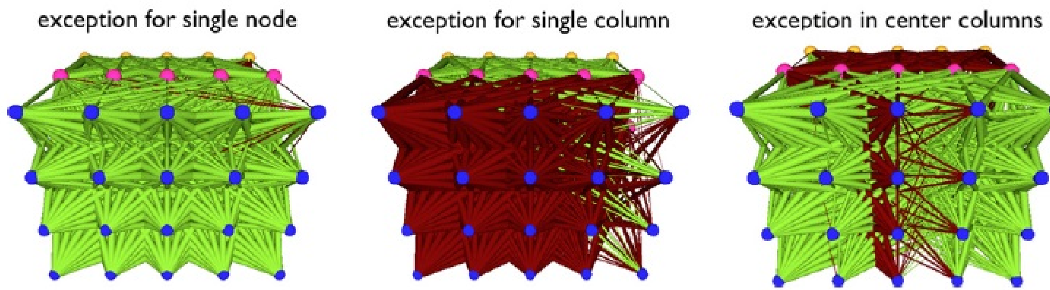
Figure 9: **Fracture in Evolved CPPN-Encoded Visual Fields.** This visualization, reproduced including the figure caption from Coleman [16], exhibits fracture in every visual field. This fracture is similar to the pattern hypothesized problematic for CPPNs in van den Berg and Whiteson [51].

3.2 Fracture in Two-Dimensional Spatial Patterns

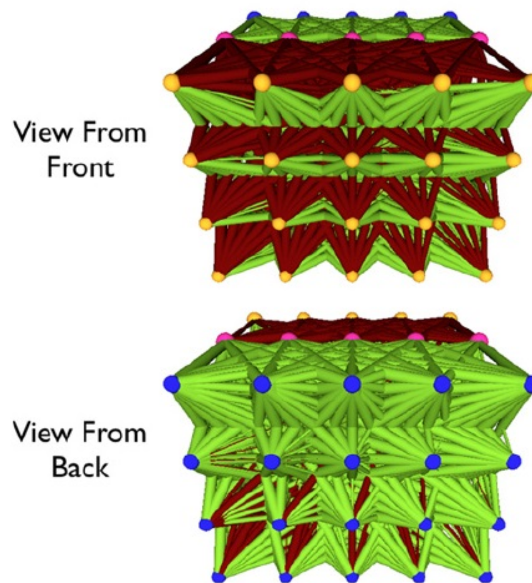
A convenient source for studying the kinds of patterns encoded by evolved CPPNs is Picbreeder [45, 46], an online service (http://picbreeder.org) where users have interactively evolved CPPN-encoded two-dimensional patterns for the last six years. Over 9,000 such patterns have been published on the site, all of which are publicly available for observation and analysis through the Picbreeder DNA tool. The existence of this large public repository of CPPN-encoded patterns presents an ideal opportunity for learning the properties of patterns encoded by CPPNs. A sampling of patterns from Picbreeder (figure 11) gives a sense of the widespread presence of fracture in CPPN-generated patterns.

While the visual appearance of fracture in output patterns provides evidence for its routine presence, its underlying representation within the CPPN further elucidates how it originates. It turns out that fractured regions originate deeply within the multilayer CPPNs that encode them, where it is possible to identify individual nodes responsible for different regions. For example, observe the *Apple* pattern in figure 12. The Apple is a good example of fracture because it includes both a fractured asymmetric stem region and a symmetric body region. Thus somehow the encoding must separate the asymmetric area from the symmetric area. The CPPN that encodes the Apple has 83 hidden nodes and 264 connections; it is the result of 320 generations of interactive evolution.

Because van den Berg and Whiteson [51] hypothesize that the number of nodes necessary to represent



(a) Simple Fractures



(b) More Complex Fracture

Figure 10: **Fracture in Evolved CPPN-Encoded Three-Dimensional Networks (reproduced from Clune et al. [15]).** Three-dimensional weight patterns in (a) exhibit non-periodic *exceptions* of different types. The fracture in the network in (b), which is shown from front and back, is more complex; a periodic pattern in the front is fractured from an entirely different pattern in the back. Note that this example demonstrates *periodic fracture*, which is a complex kind of fracture discussed by van den Berg and Whiteson [51].

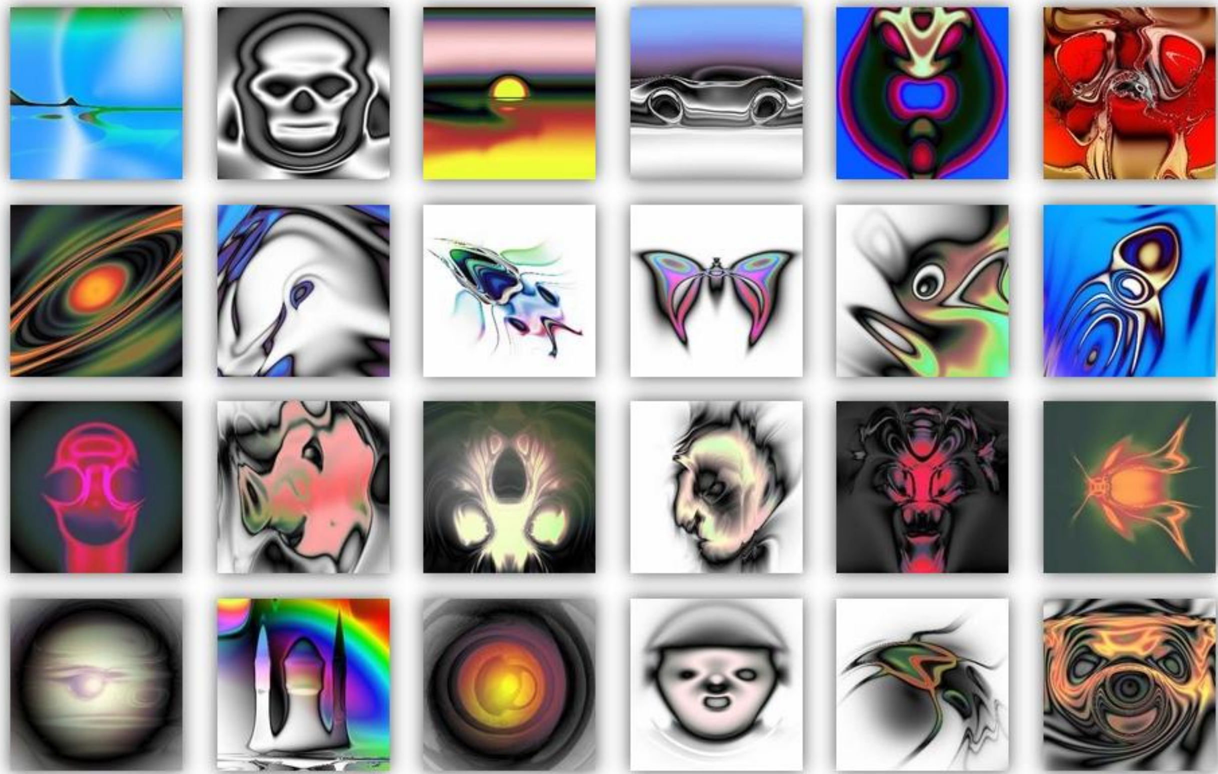


Figure 11: **Fractured Patterns Generated by CPPNs from Picbreeder.** Fracture is predominant among two-dimensional CPPN-encoded patterns evolved on Picbreeder.

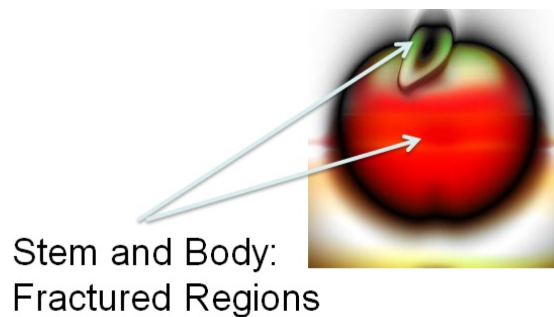


Figure 12: **Fracture in the Apple.** The stem and the body are two separately-represented fractured regions in this CPPN-encoded pattern.

fracture presents a challenge, it is informative to investigate the internal representation of fracture in patterns like the Apple. In Picbreeder the CPPN for any pattern on the site can be observed by pressing the "DNA" button under that image, which opens the visualization interface shown in figure 13. Because the Apple CPPN is very large, it is difficult to view all at once. Figure 14 gives a sense of its size.

Notice that in the Picbreeder DNA tool it shows the pattern output by *every node* within the CPPN.

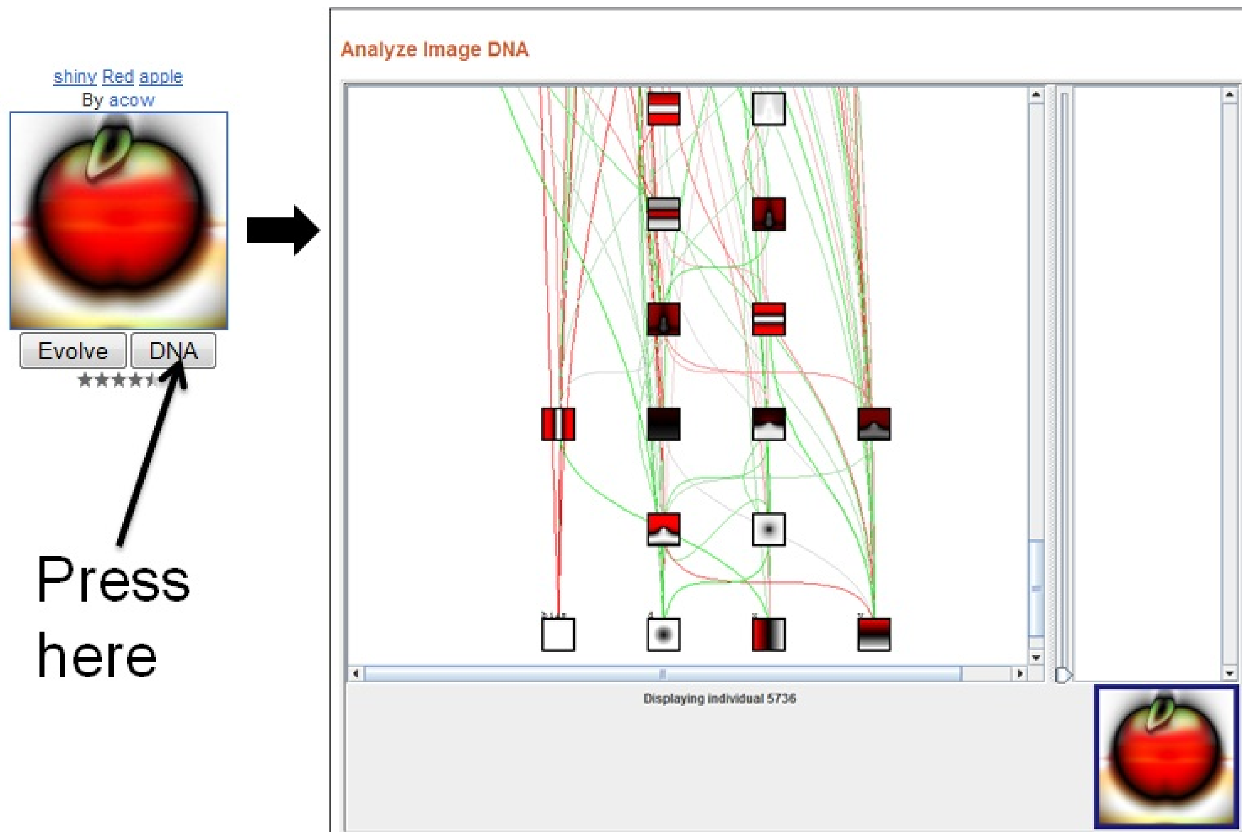


Figure 13: **Visualizing the Apple CPPN.** Picbreeder allows the user to visualize the underlying CPPN representation of any image on the site by pressing the DNA button.

That way, it becomes possible to determine the precise locations within the network where specific shapes and regions originate. In effect, the visualization shows how the final pattern is composed hierarchically within the representation from bottom (inputs) to top (outputs). Using this capability, the origins of both the fractured Apple body region and the asymmetric stem region can be identified deep within the network (figure 15). This visualization is important because it shows why CPPNs are naturally suited to representing fracture: The independent region that will encompass the main apple body (shown at left in figure 15) is seen encoded by a separate node than the one that encodes the asymmetric stem region (upper right).

The node identified as the "main source of asymmetry in apple stem" in figure 15 is interesting because its asymmetric gradient provides the asymmetry that ultimately structures the stem region. That is, the stem is a function of this deeper gradient within the CPPN, while the body region is not a function of (i.e. is not connected from) this asymmetric region. In fact, the contribution of the asymmetric gradient can be proven by *knocking out* the node that encodes it, similarly to a gene knockout experiment in biology (e.g. Baba et al. [4]). Figure 16 shows the effect on the final output pattern when the asymmetry node is deleted from the CPPN (including deleting all its incoming and outgoing connections). Interestingly, *only* the stem

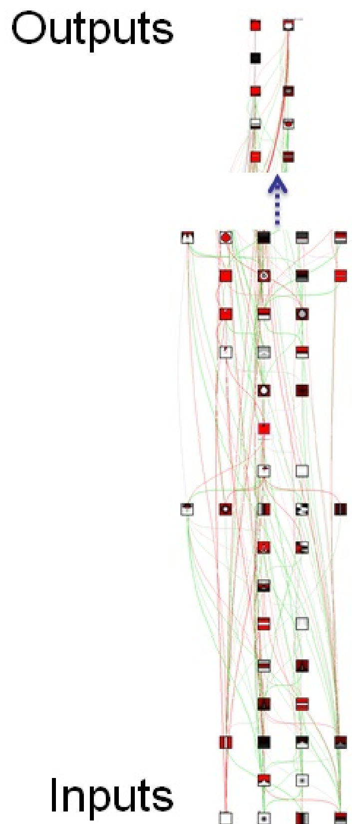


Figure 14: **Apple CPPN from Inputs to Outputs.** This visualization gives a sense of the size and structure of the 83- hidden- node and 264-connection Apple CPPN. Note that some structure is omitted (at the location of the dotted line) to save space.

is affected, which loses most of its asymmetry, demonstrating the depth of fracture in the CPPN. It is also interesting that the CPPN divides the Apple into these two regions (stem and body) when those are also intuitively the two regions that humans would identify in the image as well.

The kind of fundamental fracture seen deep within the representation of the Apple appears in many other CPPN-encoded patterns on Picbreeder. For example, the *Car* CPPN (figure 17) includes separate masks for the body, roof, and wheel orbits, which thereby constitute three intuitive fractured regions.

Another example of intuitive fracture is the *Skull* pattern. As shown in figure 18, the Skull contains masks both for its mouth region and its overall head region. To illustrate once again how deep and intuitive the fracture is within this pattern, observe the single connection highlighted in figure 18 with the label "notice this connection." That connection is important because it feeds directly into the node that defines the fractured mouth region. In effect, it is like a gene representing the mouth. Consequently, the *weight* of that single connection can be perturbed to observe the effect on the final output pattern (figure 19). Confirming

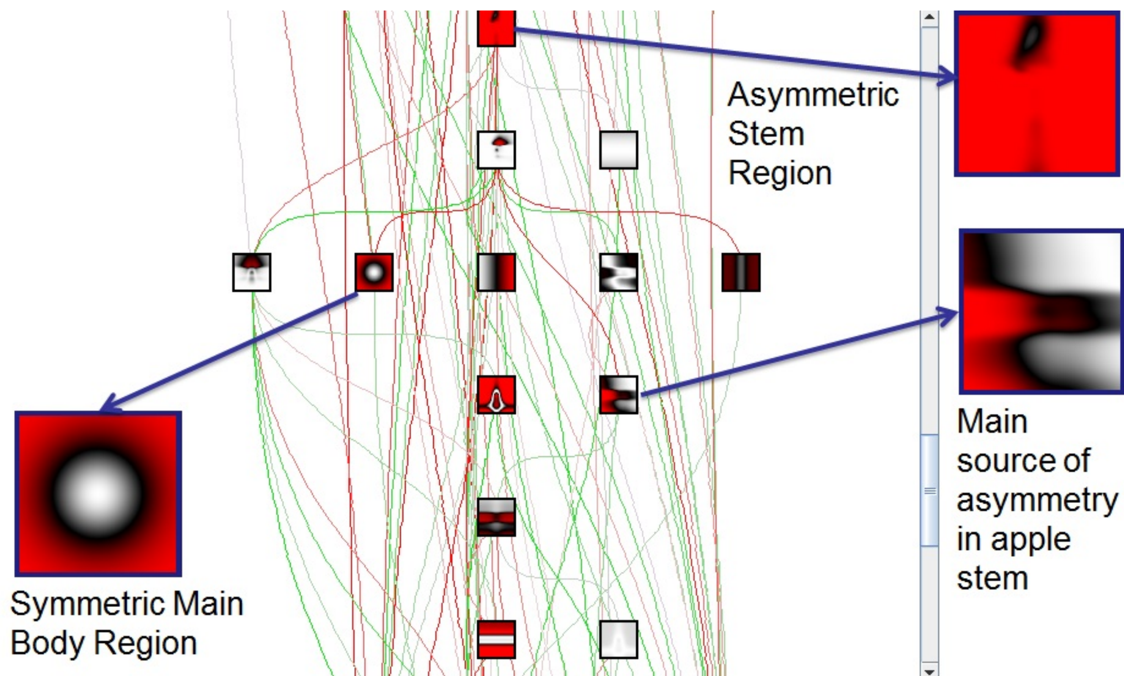


Figure 15: **Origin of Fractured Regions within the Apple.** The origin of the main symmetric body region and the asymmetric stem region are both identified. Another node is also highlighted that provides the primary asymmetry for the stem region.

once again the depth of the fracture, changing the weight of this single connection impacts only the mouth of the skull, which can be caused to open by slightly increasing the weight. Thus, surprisingly, this gene on its own opens and closes the mouth of the Skull.

Another interesting aspect of these kinds of "mask" nodes that underlie fracture is that they are sometimes conserved throughout lineages. That is, even as a genetic lineage evolves, the underlying mask pattern deep within the network is preserved with minor variation. For example, a *proto-face* mask can be observed within every step of the lineage of faces shown in figure 20. Confirming its critical contribution to the ultimate face patterns that are output, knocking it out completely removes any face-like details from the resultant output (figure 20, bottom right).

CPPNs can also represent highly complex fractured regions; not only can the regions be separated, but they can contain their own complex patterns. Fracture can also exhibit symmetry (such as when one fractured region is a reflection of another), regularity, and embedding (which means a fractured region can be embedded within a larger set of contours). Figure 21 shows several such kinds of complex fracture in CPPN-encoded patterns.

The evidence presented in this section from two-dimension patterns encoded by CPPNs demonstrates the prevalence of fracture in CPPN-encoded patterns. Thousands more examples are at http://picbreeder.org.

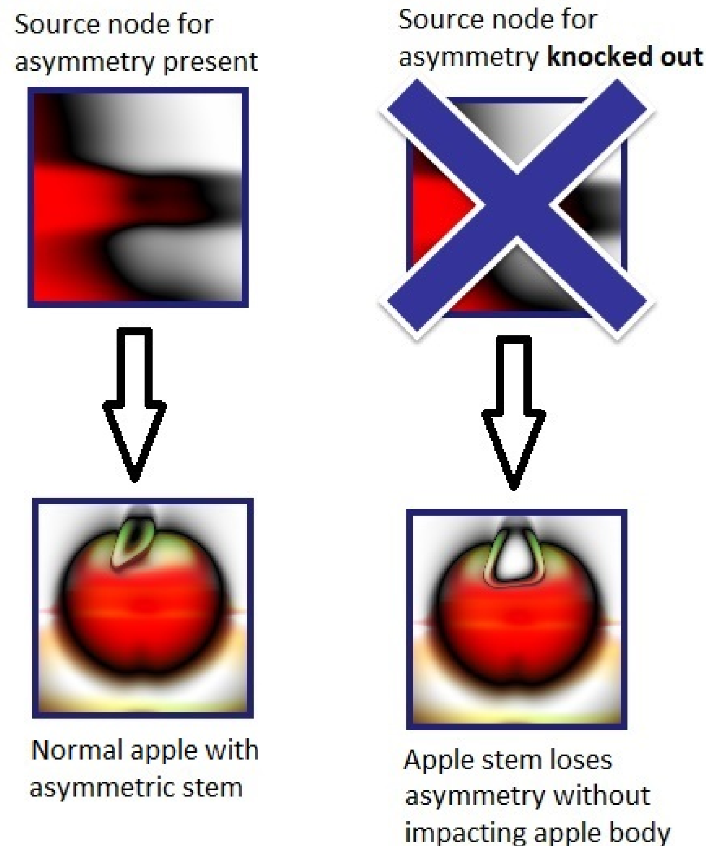


Figure 16: **Gene Knockout in the Apple.** The main asymmetry node (left) contributing to the Apple stem is knocked out (right), causing the stem to lose most of its asymmetry. Meanwhile, the rest of the Apply body remains unchanged. The conclusion is that the stem is a genuine fractured region encoded largely independently from the body region.

In fact, rather than posing a problem for CPPNs, fracture is ubiquitous. Almost every CPPN-encoded pattern exhibits some form of fracture and much of it is encoded by individual masks for each fractured region deep within the network. In this way, the Picbreeder collection offers a unique opportunity to learn about the encoding of fracture within evolved patterns.

3.3 Fracture in CPPN-encoded Morphologies

Researchers in recent years have begun to encode the morphologies of robots and artificial creatures with CPPNs [3, 9, 44]. These offer an additional opportunity to observe fracture in CPPN-encoded patterns. Some examples are shown in figure 22.

Another rich source of examples of fracture in CPPN-encoded three-dimensional objects is Endless-Forms.com [12], which is similar to Picbreeder but with three-dimensional objects instead of two-dimensional.

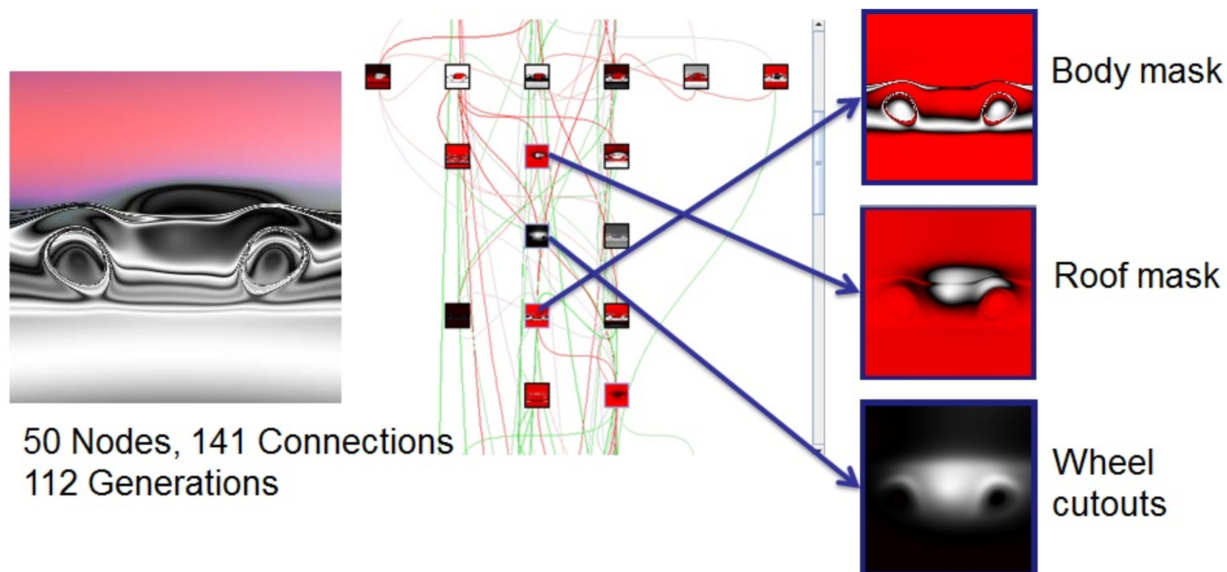


Figure 17: **Fracture in the Car CPPN.** Masks from nodes within the CPPN for three of the fractured regions in this pattern are shown at right.

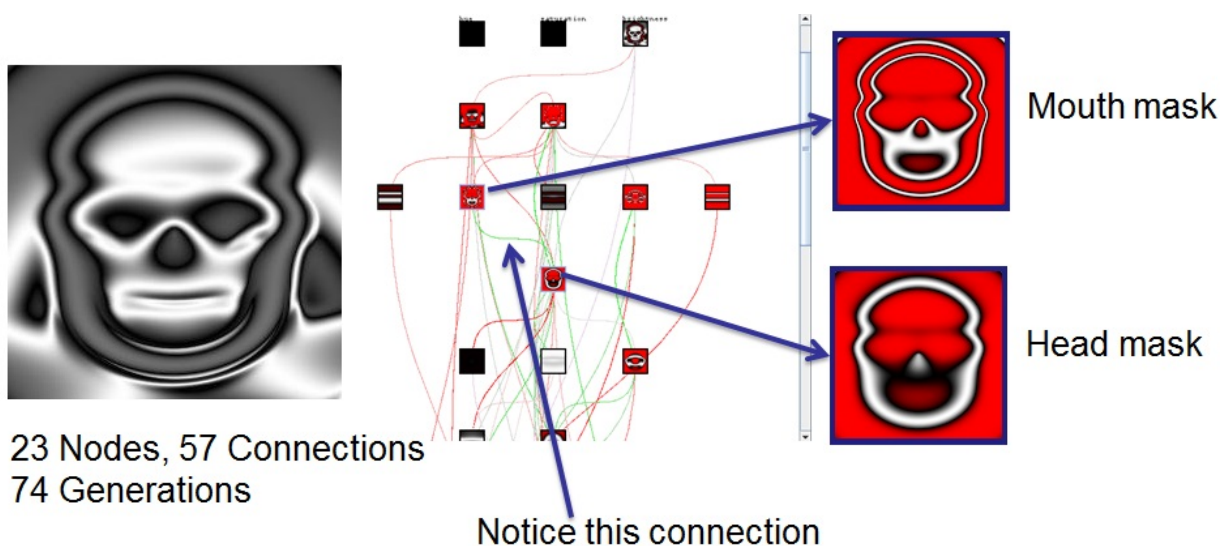


Figure 18: **Fracture in the Skull CPPN.** Masks from nodes within the CPPN for the mouth and head regions are shown at right.

Figure 23 shows examples of fracture in three-dimensional objects from EndlessForms generated by CPPNs.

Taken together, the visualizations of fracture within evolved HyperNEAT networks, two-dimensional CPPN-encoded images, and CPPN-encoded three-dimensional robots and other objects highlight the ubiquity of fracture across CPPN-encoded artifacts and the considerable nuance and subtlety achievable through CPPNs. Studies of the internal nodes of such CPPNs further show that fracture is often encoded deeply within evolved networks, separating surprisingly intuitive regions like stems and bodies and mouths and

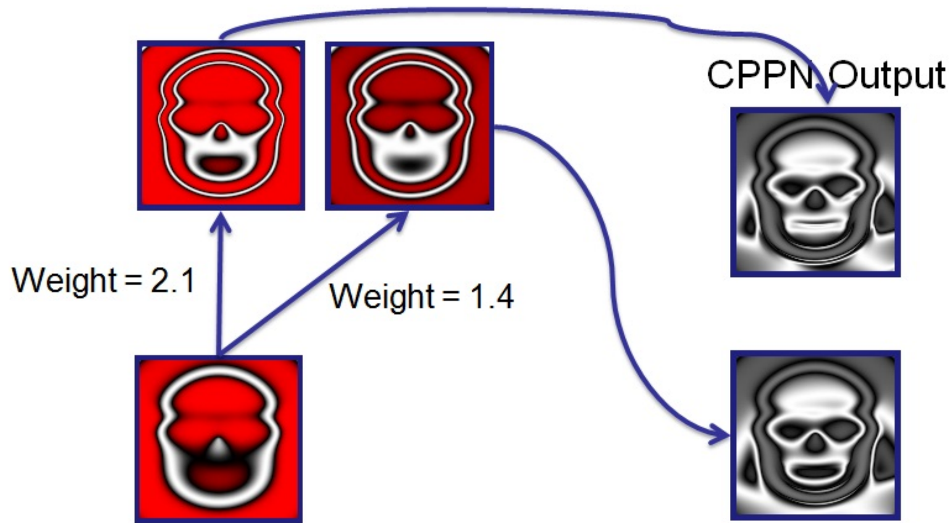


Figure 19: **Opening and Closing the Skull's Mouth with a Single Connection Weight.** Changing the weight of the single connection noted in figure 18 determines the degree to which the Skull's mouth is open. In this way the mouth is a genuine fractured region with its own independent parameters within the evolved CPPN.

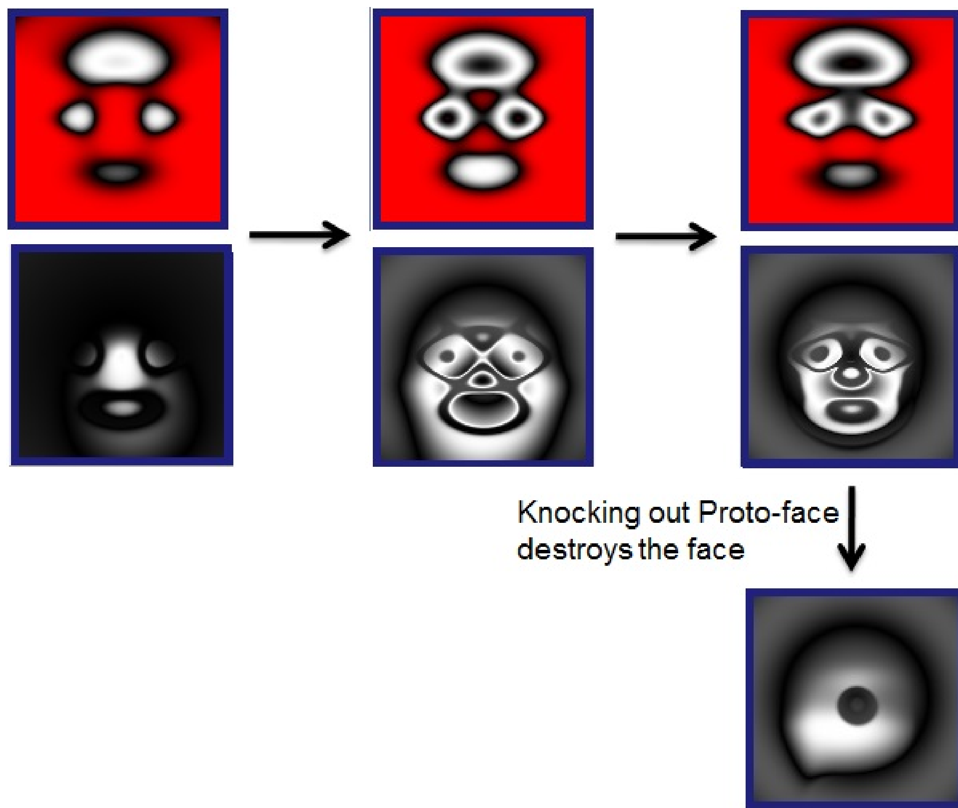
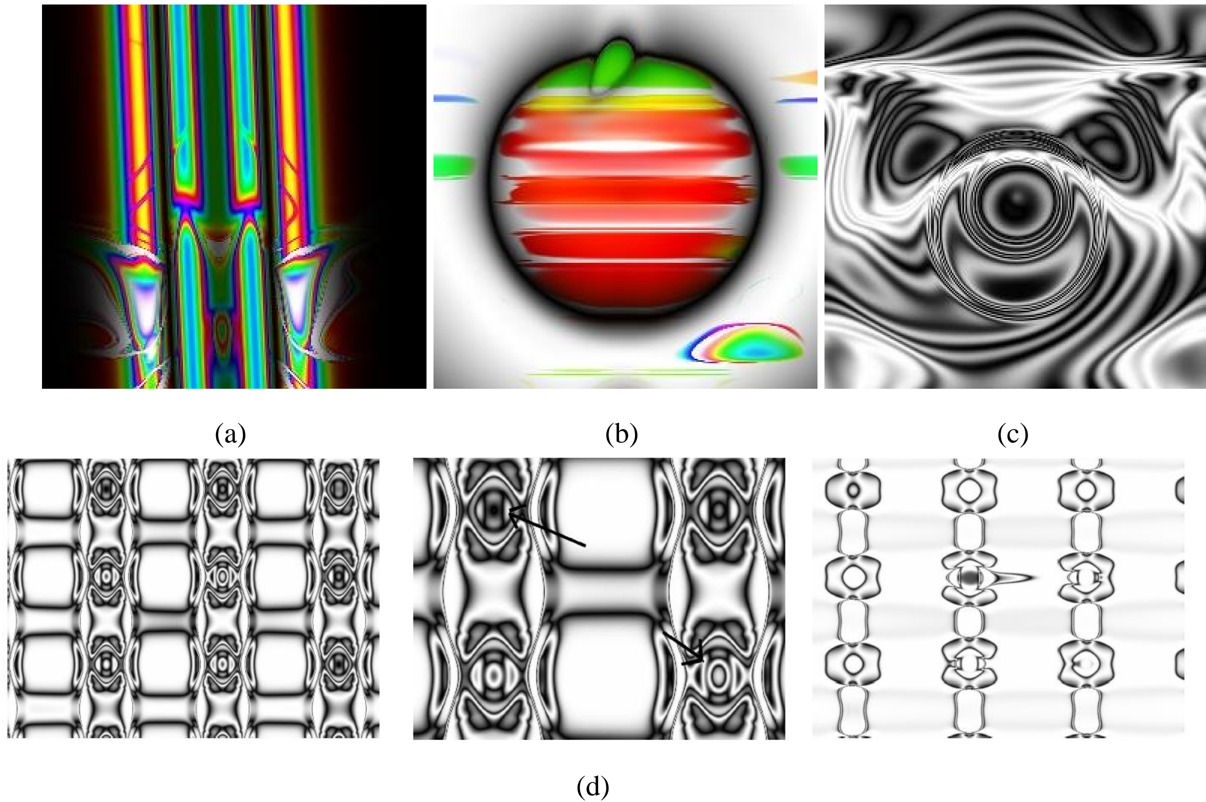


Figure 20: **Conserved Proto-Face Mask throughout the Face Lineage.** The masks (top) are patterns encoded by internal nodes from each of the faces in the lineage below. As the lineage evolves from left to right, the proto-face mask is conserved. Knocking out the mask destroys the face (bottom right), confirming the importance of the mask to ultimately forming the pattern of a face.


(a) (b) (c)

(d)

Figure 21: **Examples of Complex Fracture.** In the pattern in (a), each region contains its own nontrivial color pattern, some of which are reflections of others. The fractured region enclosed by the apple variant in (b) contains a complex "brush-stroke" pattern that differs markedly from the pattern in the outer fractured region. The complex fractured regions in (c), which contain their own periodic patterns, are layered upon a distorted surface contour, yielding a three-dimensional effect. The CPPN-generated patterns in (d) (reproduced from Stanley [48]) exhibit repeated fracture with nested fracture. The middle frame is a zoomed version of the left frame, with an arrow highlighting that not only is the fractured pattern repeated, but it is repeated with variation.

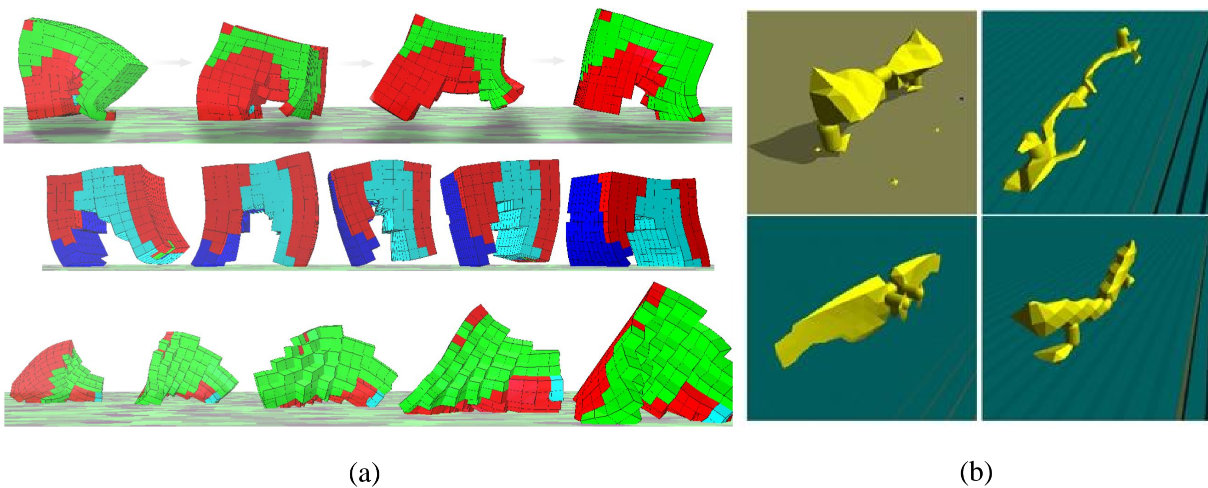

(a) (b)

Figure 22: **Examples of Fracture in CPPN-encoded Creature Morphologies.** Fracture is visible in these reproduced images of the morphologies of soft robots evolved by Cheney et al. [9] and creatures evolved by Auerbach and Bongard [3].

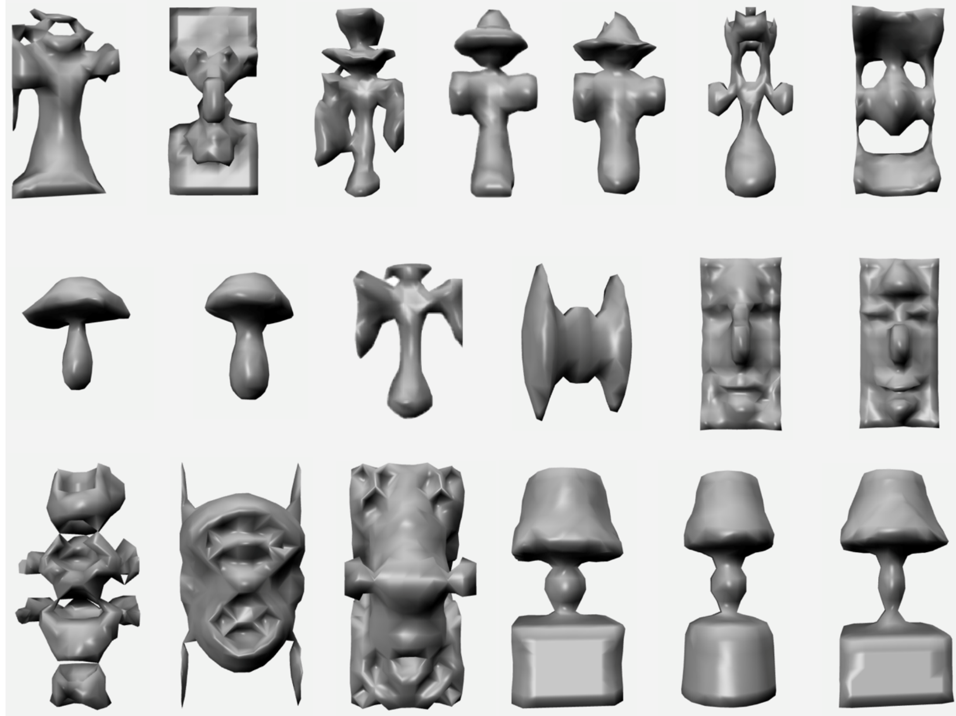


Figure 23: **Examples of Fracture from EndlessForms (from Clune and Lipson [12]).** All objects at EndlessForms.com are evolved by online users with CPPNs. Examples of fracture above include eyes, noses, and the balls within the lamp-stands.

heads. Overall, CPPNs emerge not only as a viable choice for encoding fracture, but as a promising platform for studying and learning about fracture in general.

Although fracture is evident in the target-driven domains shown in Sections 3.1 and 3.3, it is also interesting that fracture is seen in such abundance in tools like Picbreeder and EndlessForms, in which there is no explicit fitness-based target. The degree to which fracture emerges under strictly target-based conditions versus more open-ended conditions is an interesting question for future research. Methods for preserving behavioral diversity such as novelty search [31, 34] and others [33, 40, 42, 54] provide a potential path to endowing practical domains with a more open-ended notion of fitness, which may increase the ability of neuroevolution methods to respond with fractured solutions.

4 Discussion

This section first revisits the fracture hypothesis in light of the results reported in this paper and then discusses deeper issues in benchmark performance in the field of generative and development systems (GDS).

4.1 Revisiting the Fracture Hypothesis

The results in the two reimplemented domains suggest that HyperNEAT indeed *can* solve tasks that van den Berg and Whiteson [51] found it could not. In particular, HyperNEAT does solve the triangles visual discrimination problem by taking full advantage of hidden nodes and it does solve the line following problem with distractor bands. While van den Berg and Whiteson [51] hypothesize based on their results that "fracture can be highly problematic for HyperNEAT," these newer results suggest that the critical factor in HyperNEAT is in fact *not* fracture. Given this discrepancy between the findings here and those of van den Berg and Whiteson [51], several lessons can be learned from drawing implications from such experiments about the real factors that impact performance:

- All the factors that enter into a complicated system like HyperNEAT may matter, including the implementation of the algorithm, the parameters of the system, the setup of the task domain and reward function, and resources (such as population size) given to the algorithm. Thus it can be difficult sometimes to disentangle which factors are critical and which are not.
- Sometimes a change in a system parameter can make a significant difference in performance. For example, raising the structural mutation rate in PEAS-NEAT was enough to show that adding CPPN hidden nodes indeed *does* provide an advantage over restricting their number to zero.
- It can take time for a system like HyperNEAT that allows increasing complexity to take advantage of that increasing complexity. SharpNEAT-based results with HyperNEAT in this paper show that in some runs the system can even eventually exhibit *perfect* training performance by taking advantage of CPPN hidden nodes, which is impossible without them.
- Experimental setup can interact with behavior in subtle ways. For example, SharpNEAT-based HyperNEAT easily solves the difficult line following task without hidden nodes, even achieving full coverage in most attempts even though it is not explicitly rewarded by the fitness function. Thus whether or not an experimental domain actually tests a particular hypothesis may depend on the precise setup of the domain. In the case of line following as described by van den Berg and Whiteson [51], the domain does not appear ultimately to address the ability to add hidden nodes in the SharpNEAT setup.

Overall, the results in the reimplemented experimental domains suggest that (1) HyperNEAT *can* take advantage of CPPN hidden nodes, including in a case (triangles) where it seems the only way to achieve the best performance on the task is with hidden nodes. (2) HyperNEAT can effectively solve both triangles

and line following. (3) HyperNEAT is also able to solve a complicated quadruped-walking task in separate work [41, 43], though it is not exactly the same problem as attempted by van den Berg and Whiteson [51]. These outcomes diminish the motivation for the hypothesis that HyperNEAT has a problem with fracture and also suggest it does not have a problem with taking advantage of hidden nodes. Positive evidence that CPPNs can and do produce fracture is provided in the second half of this paper (Section 3), which shows numerous examples of CPPN-encoded fracture in substrates, in two-dimensional output patterns, and in robot morphologies. This survey of evidence suggests in fact that fracture with evolved CPPNs is the norm rather than the exception, and furthermore that much remains to be learned from understanding the representation of such fracture. In fact, few other extensive such demonstrations of fracture in evolved patterns exist for any encoding other than CPPNs.

In conclusion, the factors contributing to the results of van den Berg and Whiteson [51] are likely not fracture-related, and HyperNEAT can solve such problems without significant difficulty.

4.2 Interpreting Benchmark Performance in GDS

While HyperNEAT can solve the two reimplemented problems from van den Berg and Whiteson [51], of course there will be other problems that various implementations of HyperNEAT will still struggle to solve. What meaningful inferences can be drawn from such failures when they arise?

Interpreting the implications of failures is potentially more tricky than interpreting successes. When an experiment succeeds, there is no ambiguity that the method in question can solve the tested problem. Of course, it may not *always* succeed at that problem, but at least it is accurate to conclude that it *can* succeed. However, the converse is not true for failures: If a method fails to solve a problem, it does not follow that it *cannot* solve the problem. It is possible that it actually can solve the problem, but under slightly different conditions, or with a slightly different implementation. The problem is that proving a negative requires substantially more evidence than proving a positive. Therefore, negative results merit significant caution in their general interpretation.

A particularly relevant example of this danger is given in Woolley and Stanley [53], which investigates the ability of the NEAT algorithm to evolve CPPNs that output patterns that match particular Picbreeder images (such as those in figure 11). The result there is that NEAT fails to reproduce the target images in all but the most trivial cases. Under normal circumstances, this result would be unremarkable, and many would conclude simply that for whatever reason NEAT cannot evolve CPPNs that encode the target images. From there it would be possible to attempt to infer a hypothesis to explain the failure. Perhaps even a problem

with evolving fracture could be hypothesized (considering that so many Picbreeder images exhibit fracture).

However, the circumstances of the experiment in Woolley and Stanley [53] are not normal because of one critical caveat: NEAT already *did* evolve the images in Picbreeder. After all, NEAT is the algorithm that evolves the CPPNs in Picbreeder. Therefore, in this case, whether or not NEAT could evolve these images in the particular experiment of Woolley and Stanley [53], the conclusion that NEAT "cannot" evolve such images is not viable. Not only can NEAT evolve such images, but NEAT is the only algorithm ever to evolve any of them. This contradiction highlights the danger of drawing general inferences from negative results.

Yet with respect to the conclusions in van den Berg and Whiteson [51], the implications for the interpretation of negative results are even deeper because van den Berg and Whiteson [51] attempt to validate their fracture hypothesis with experiments almost identical to the target-based pattern-matching experiments in Woolley and Stanley [53]. In particular, they conclude their paper by aiming to confirm that HyperNEAT has a problem with fracture by testing HyperNEAT's ability to replicate arbitrary fractured target patterns of weights. As shown in Woolley and Stanley [53], such an experiment can be expected to fail even though CPPNs *can* evolve such fractured patterns.

Thus the question shifts to why HyperNEAT or CPPNs would fail to reproduce patterns that they are demonstrably able to produce under other conditions. A growing body of evidence [31, 32, 33, 34, 35, 36, 37, 42, 53, 54] suggests that often evolutionary algorithms fail to achieve desired results that are set as explicit objective *targets* even though the very same algorithms can succeed at the same problems when the desired result is not set as an explicit target. In other words, if the fitness function is little more than *distance to the target pattern* or *distance from optimal performance*, such algorithms tend to fail, but changing the fitness function to be less target-driven can entirely change the result.

While a detailed treatment of the problem with target-based objectives is beyond the scope of this paper (a number of sources cover the issue in detail [34, 36, 53]), an intuitive explanation is that setting the objective as an explicit target is often highly susceptible to deception. For example, if the target pattern is the Skull (figure 18), then a mutation that places the right color in *one* of the two eyes would cause an increase in fitness. However, such an increase is deceptive in the long run because the representation has failed to discover the important underlying regularity of *symmetry* that underlies the entire Skull. Thus from that moment onward, any mutation that changes one eye will not change the other side of the face accordingly, reducing the probability of reproducing the whole image correctly.

At the same time, if symmetry *is* discovered with a target-based objective, it will nevertheless not be rewarded unless it happens to look reminiscent of the Skull right away, even though the most important

fundamental discovery that could be made to set the search for the Skull in the right direction *is* bilateral symmetry.

The same kind of deception arises with behavioral performance objectives. For example, if the objective is to evolve a quadruped to walk the longest distance possible (i.e. the target is maximal distance), then an early mutation that leads a single forceful lunge of the body will be rewarded for propelling the body farther than in the past. However, deceptively again, such a large lunge is not a good stepping stone to walking because walking requires *oscillations*, not isolated lunges.

At the same time, similarly to with the target-based image, if oscillation *is* discovered, it will nevertheless not be rewarded initially unless it happens by coincidence also to be paired with good balance (which is unlikely), even though oscillation is a critical and fundamental discovery in the search for locomotion. In effect, target-based fitness functions inadvertently favor short-term gain over long-term benefit, leading to deception and premature convergence.

Moreover, there are many good alternatives, such as novelty search [34], minimal criteria novelty search [33], collaborative interactive evolution (which tends to lack a unified target objective) [12, 26, 27, 45, 46], combinations of novelty search with interactive evolution [54], and behavioral diversity techniques [40, 42]. These alternatives share the idea that evolution is rewarded not merely for distance to an objective, but for maintaining a healthy behavioral diversity. In this way, for example, discovering oscillation *can be* rewarded because it is recognized as novel whether or not it provides an immediate boost in target performance. Symmetry can be rewarded because it is interesting in its own right, regardless of whether it initially raises or lowers fitness. Some such techniques rely on measuring novelty automatically, while others include human judgment in the selection process. Still others combine novelty with objectives, such as through multiobjective optimization [42]. The main point is that it *is* possible to formulate problems through overall non-objective (i.e. less target-based) reward schemes, whereupon methods that otherwise might be hypothesized to have a problem with certain domains actually prove successful.

This observation is particularly important for the field of generative and developmental systems (GDS), which focuses on indirect encoding [1, 6, 7, 20, 28, 38, 39, 47, 48, 50]. (CPPNs are one kind of indirect encoding.) Indirect encodings rely on the discovery of compositional regularities. For example, a system might first discover symmetry and then elaborate that symmetry into four legs, two of which might later elaborate into arms. Each step in such a process of elaboration requires refining a regularity established in a previous step, that is, regularities are *composed* hierarchically over generations to achieve increasingly refined phenotypes and behaviors. Target-based fitness functions are antithetical to such a process because

they are not sensitive to anything but the final solution, thereby entirely missing (and thereby penalizing) critical regularities that should be discovered along the way. Symmetry on its own may not initially look like a face, but it is essential to capturing the essence of facial structure at a deep level. On the other hand, behavioral diversity tends to reward such discoveries in their own right, thereby accumulating promising stepping stones that can become the basis of solutions. In this way, target-based fitness functions are generally not good tests of the abilities of algorithms, but this is especially true for indirect encodings, due to the circuitous path they often need to follow to reach a solution. Thus, target-based fitness functions do not test what indirect encodings ultimately can or cannot do; rather they only tell us something about how such encodings might perform under target-based conditions.

For these reasons, drawing broad inferences on the capabilities of GDS techniques from the results of target-based problems is potentially misleading and even dangerous for the field of GDS and indirect encoding. If indirect encoding produces its most impressive results under non-objective conditions (such as in Picbreeder or even on Earth, where there is no final target for evolution), then formulating hypotheses about the capabilities of such encodings based on their performance under strictly target-based conditions will lead us inevitably astray. We will end up with seemingly contradictory conclusions (which would be confusing), such as that fracture can be problematic for HyperNEAT on the one hand, while on the other hand fracture is the essence of almost every pattern that CPPNs produce. Similarly, we would not want to conclude that DNA cannot evolve complex organisms simply because we might fail to breed a single cell into a fish or a bird.

In this way, it is important within our discourse as a field to separate carefully what an evolutionary method *can* do and what it tends to do under strictly target-based conditions.

5 Conclusion

The evidence in this paper contradicts the hypothesis that HyperNEAT and CPPNs have a problem with fracture and show it effectively utilizing hidden nodes. The results show that HyperNEAT is capable of effectively solving the two reimplemented problem domains from van den Berg and Whiteson [51] after all. It has also exhibited positive results in separate publications in domains similar to the walking gait task [41, 43] that was not reimplemented for this paper. Furthermore, this paper provides a range of examples wherein CPPNs exhibit fracture. Perhaps a deeper lesson is that evolutionary algorithms are notoriously difficult to characterize through benchmark results and the implications of such benchmarks need to be

considered carefully. That is, we need to be careful about drawing broad conclusions about interesting algorithms only from target-based benchmarks. By digging deeper (such as in the examples of fracture in this paper), we can potentially learn a tremendous amount from algorithms and encodings beyond only such superficial benchmark performance results.

Acknowledgments

Special thanks Jeremiah T. Folsom-Kovarik for programming the Picbreeder DNA tool, which provided important evidence of fracture in this paper.

References

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

[2] Auerbach, J., and Bongard, J. (2010). Evolving CPPNs to grow three dimensional structures. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)*. New York, NY: ACM Press.

[3] Auerbach, J. E., and Bongard, J. C. (2012). On the relationship between environmental and mechanical complexity in evolved robots. In *Proceedings of 13th International Conference on the Synthesis and Simulation of Living Systems (ALife XIII)*.

[4] Baba, T., Ara, T., Hasegawa, M., Takai, Y., Okumura, Y., Baba, M., Datsenko, K. A., Tomita, M., Wanner, B. L., and Mori, H. (2006). Construction of escherichia coli k-12 in-frame, single-gene knockout mutants: the keio collection. *Molecular systems biology*, 2(1).

[5] Bahceci, E., and Miikkulainen, R. (2008). Transfer of evolved pattern-based heuristics in games. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG-2008)*. Piscataway, NJ: IEEE Press.

[6] Bentley, P. J., and Kumar, S. (1999). Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, 35–43. San Francisco: Kaufmann.

[7] Bongard, J. C. (2002). Evolving modular genetic regulatory networks. In *Proceedings of the 2002 Congress on Evolutionary Computation*.

[8] Buk, Z., Koutnk, J., and Snorek, M. (2009). NEAT in HyperNEAT substituted with genetic programming. In *Inernational Conference on Adaptive and Natural Computing Algorithms (ICANNGA-2009)*, 243–252. Berlin: Springer.

[9] Cheney, N., MacCurdy, R., Clune, J., and Lipson, H. (2011). Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-20011)*. New York, NY: ACM Press.

[10] Clune, J., Beckmann, B., McKinley, P., and Ofria, C. (2010). Investigating whether HyperNEAT produces modular neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2010)*. New York, NY: ACM Press.

[11] Clune, J., Beckmann, B. B., Pennock, R., and Ofria, C. (2009). HybrID: A hybridization of indirect and direct encodings for evolutionary computation. In *Proceedings of the European Conference on Artificial Life (ECAL-2009),*.

[12] Clune, J., and Lipson, H. (2011). Evolving three-dimensional objects with a generative encoding inspired by developmental biology. In *Proceedings of the European Conference on Artificial Life (ECAL-2011)*, 141–148.

[13] Clune, J., Ofria, C., and Pennock, R. (2008). How a generative encoding fares as problem-regularity decreases. In *Proceedings of the 10th International Conference on Parallel Problem Solving From Nature (PPSN 2008)*, 258–367. Berlin: Springer.

[14] Clune, J., Pennock, R. T., and Ofria, C. (2009). The sensitivity of HyperNEAT to different geometric representations of a problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2009)*. New York, NY, USA: ACM Press.

[15] Clune, J., Stanley, K. O., Pennock, R. T., and Ofria, C. (2011). On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation*.

[16] Coleman, O. (2010). Evolving neural networks for visual processing. Undergraduate Thesis, University of New South Wales School of Computer Science and Engineering.

[17] D'Ambrosio, D., Lehman, J., Risi, S., and Stanley, K. O. (2010). Evolving policy geometry for scalable multiagent learning. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2010)*, 731–738. International Foundation for Autonomous Agents and Multiagent Systems.

[18] Drchal, J., Kapra, O., Koutnik, J., and Snorek, M. (2009). Combining multiple inputs in HyperNEAT mobile agent controller. In *19th International Conference on Artificial Neural Networks (ICANN 2009)*, 775–783. Berlin: Springer.

[19] Drchal, J., Koutnk, J., and Snorek, M. (2009). HyperNEAT controlled robots learn to drive on roads in simulated environment. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2009)*. Piscataway, NJ, USA: IEEE Press.

[20] Eggenberger, P. (1997). Evolving morphologies of simulated 3D organisms based on differential gene expression. In Husbands, P., and Harvey, I., editors, *Proceedings of the Fourth European Conference on Artificial Life*, 205–213. Cambridge, MA: MIT Press.

[21] Gauci, J., and Stanley, K. O. (2008). A case study on the critical role of geometric regularity in machine learning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008)*. Menlo Park, CA: AAAI Press.

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

[23] Gomez, F., Koutnik, J., and Schmidhuber, J. (2012). Compressed network complexity search. In *Proceedings of the 12th International Conference on Parallel Problem Solving from Nature (PPSN XII)*.

[24] Green, C. (2003–2006). SharpNEAT homepage. `http://sharpneat.sourceforge.net/`.

[25] Haasdijk, E., Rusu, A. A., and Eiben, A. (2010). HyperNEAT for locomotion control in modular robots. In *Proceedings of the 9th International Conference on Evolvable Systems (ICES 2010),*.

[26] Hastings, E. J., Guha, R. K., and Stanley, K. O. (2009). Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):245–263.

[27] Hastings, E. J., Guha, R. K., and Stanley, K. O. (2009). Interactive evolution of particle systems for computer graphics and animation. *IEEE Transactions on Evolutionary Computation*, 13(2):418–432.

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

[29] Jaskowski, W., Krawiec, K., and Wieloch, B. (2008). Neurohunter - an entry for the balanced diet contest. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2008) Contest Program")*. New York, NY: ACM Press.

[30] Kohl, N., and Miikkulainen, R. (2009). Evolving neural networks for strategic decision-making problems. *Neural Networks, Special issue on Goal-Directed Neural Systems*.

[31] Lehman, J., and Stanley, K. O. (2008). Exploiting open-endedness to solve problems through the search for novelty. In Bullock, S., Noble, J., Watson, R., and Bedau, M., editors, *Proceedings of the Eleventh International Conference on Artificial Life (Alife XI)*. Cambridge, MA: MIT Press.

[32] Lehman, J., and Stanley, K. O. (2010). Efficiently evolving programs through the search for novelty. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, 837–844. New York, NY, USA: ACM.

[33] Lehman, J., and Stanley, K. O. (2010). Revising the evolutionary computation abstraction: minimal criteria novelty search. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, 103–110. New York, NY, USA: ACM.

[34] Lehman, J., and Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223.

[35] Lehman, J., and Stanley, K. O. (2011). Evolving a diversity of virtual creatures through novelty search and local competition. In *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 211–218. Dublin, Ireland: ACM.

[36] Lehman, J., and Stanley, K. O. (2011). Novelty search and the problem with objectives. In *Genetic Programming Theory and Practice IX (GPTP 2011)*. New York, NY: Springer.

[37] Lehman, J., and Stanley, K. O. (2012). Beyond open-endedness: Quantifying impressiveness. In *Proceedings of the Thirteenth International Conference on Artificial Life (ALIFE XIII)*. Cambridge, MA: MIT Press.

[38] Lindenmayer, A. (1974). Adding continuous components to L-systems. In Rozenberg, G., and Salomaa, A., editors, *L Systems, Lecture Notes in Computer Science 15*, 53–68. Heidelberg, Germany: Springer-Verlag.

[39] Miller, J. F. (2004). Evolving a self-repairing, self-regulating, French flag organism. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*. Berlin: Springer Verlag.

[40] Moriguchi, H., and Shinichi, H. (2010). Sustaining behavioral diversity in neat. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, 611–618. New York, NY, USA: ACM.

[41] Morse, G., Risi, S., Snyder, C. R., and Stanley, K. O. (2013). Single-unit pattern generators for quadruped locomotion. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2013)*. New York, NY, USA: ACM.

[42] Mouret, J.-B., and Doncieux, S. (2012). Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evolutionary computation*, 20(1):91–133.

[43] Risi, S., , and Stanley, K. O. (2013). Confronting the challenge of learning a flexible neural controller for a diversity of morphologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2013)*. New York, NY, USA: ACM.

[44] Risi, S., Cellucci, D., and Lipson, H. (2013). Ribosomal robots: Evolved designs inspired by protein folding. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2013)*. New York, NY: ACM.

[45] Secretan, J., Beato, N., D.Ambrosio, D. B., Rodriguez, A., Campbell, A., Folsom-Kovarik, J. T., and Stanley, K. O. (2011). Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evolutionary Computation*, 19(3):345–371.

[46] Secretan, J., Beato, N., D'Ambrosio, D. B., Rodriguez, A., Campbell, A., and Stanley, K. O. (2008). Picbreeder: Evolving pictures collaboratively online. In *CHI '08: Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, 1759–1768. New York, NY, USA: ACM.

[47] Sims, K. (1994). Evolving 3D morphology and behavior by competition. In Brooks, R. A., and Maes, P., editors, *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems (Artificial Life IV)*, 28–39. Cambridge, MA: MIT Press.

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

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

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

[51] van den Berg, T., and Whiteson, S. (2013). Critical factors in the performance of HyperNEAT. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2013)*. New York, NY, USA: ACM.

[52] Verbancsics, P., and Stanley, K. O. (2010). Evolving static representations for task transfer. *Journal of Machine Learning Research (JMLR)*, 11:1737–1769.

[53] Woolley, B. G., and Stanley, K. O. (2011). On the deleterious effects of a priori objectives on evolution and representation. In *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 957–964. Dublin, Ireland: ACM.

[54] Woolley, B. G., and Stanley, K. O. (2012). Exploring Promising Stepping Stones by Combining Novelty Search with Interactive Evolution. *ArXiv e-prints*.