Developmental Encodings Promote the Emergence of Hierarchical Modularity

Jessica Lowell[1] and Jordan Pollack[1]

[1]DEMO Lab, Departmentl of Computer Science, Brandeis University, Waltham, MA 02453
jessiehl@cs.brandeis.edu

Abstract

While it has been observed (Hornby et al., 2001) that developmental encodings in evolved systems may promote modularity, there has been little quantitative study of this phenomenon. There has also been little study of the factors driving the emergence of hierarchical modularity - modularity on multiple levels, in which the modules found at a finer-grained level can serve as elements in a coarser-grained network that is also modular - despite the fact that most fields with an interest in modularity, including biology and engineering, define hierarchy as an important aspect of modularity. We examine the effect of developmental encodings on the emergence of multiple levels of modularity through the lens of two developmental systems, GRNEAT and GENRE, and find evidence that developmental encodings promote this emergence of modular hierarchy.

Introduction

Below, we examine interactions between development and hierarchical modularity in artificial systems. Modularity, the organization of a system into a hierarchical system of interacting subparts, is observed in many systems both natural and engineered (Koza, 1992; Simon, 1996; Hartwell et al., 1999), and has become important as evolutionary systems are used in increasingly complex applications. Simulations of development, the process by which a mature phenotype is constructed from an organism's genetic code, have been used in computational studies both in conjunction with and distinct from simulations of evolution. We briefly discuss modularity in evolution, followed by an overview of artificial development.

Modularity

Biological systems, including biological networks such as neural networks and bacterial metabolic networks, and other kinds of biological systems such as tissues (which are assembled from cells), tend to be modular. The definition of modularity is somewhat vague - though generally referring to the degree to which a system is composed of separable, recombinable components - and can be used differently in different fields and subfields. Bolker (2000) attempted to define a list of characteristics of modularity that would be appropriate across different subfields and levels of study in biology, including greater internal integration of modules as compared to external integration, the ability to delineate modules from their surroundings, and module performance that is greater than the sum of its parts. Schilling (2002) found that a variety of fields, including technology, psychology, biology, American studies, and mathematics, define hierarchical nesting as an aspect of modularity. It is worth noting that the hierarchical aspect of modularity, the emergence of which we explore in this paper, has not traditionally been examined in simulated evolution studies, despite its importance in how most fields define modularity. We chose to focus on hierarchy because of this gap in the literature, and because development is such a key factor in the formation of many hierarchically modular biological systems, such as organisms.

Evolutionary algorithms tend to produce nonmodular solutions - though there are some exceptions, as in the coevolutionary algorithm of Juille and Pollack (1996), which used genetic programming to produce modular solutions to the intertwined spirals problem. These nonmodular solutions are often connected in complicated ways and perform better on the task for which they are optimized than the more modular solutions designed by human designers (Thompson, 2012) (Vassilev et al., 2000). However, while this tendency against modularity can produce well-performing solutions for simple problems, it makes it difficult for evolved systems to solve complex problems (Kashtan and Alon, 2005). While this issue can be addressed by building the encapsulation of modules into algorithms, this does not illuminate how modularity evolves in nature, and it means possibly missing out on some design benefit that comes with modularity emerging rather than being hard-coded. In addition, allowing modularity to emerge through an iterative process may allow for nonmodular, high-performing species of solutions to develop modularity over time while preserving their strong performance.

In recent years, there have been several studies examining the emergence of modularity in both natural and sim-

ulated evolution. (Lipson et al., 2002) found, in a study of minimal substrate modularization, that modular separation is logarithmically proportional to rates of environmental variation, and suggested using variable rather than fixed fitness criteria for the evolutionary design of engineered systems. This hypothesis was supported by the work of (Kashtan and Alon, 2005) in computational evolution studies, and (Kashtan et al., 2007; Parter et al., 2007) in natural evolution studies, which found that modularity evolves in response to varying environments (called modularly varying goals) in which individuals perform varying tasks that are decomposable into common subtasks. The requirement that subtasks be performed in sequence, as a chain, has also been found to promote the evolution of modularity (Calcott, 2014). Another possible explanation for modularity's evolution was proposed by (Clune et al., 2013), which suggested that modular networks evolve in response to a small decrease in fitness for each connection in the network - a connection cost - representing the energy cost of forming a link in a physical network. A similar energy cost imposed on the NEAT neuroevolution algorithm, on a problem in which some solutions that evolve are modular, has been found to increase consistency in modularity emergence (Lowell and Pollack, 2014).

Artificial Development

Artificial development, also known as artificial embryology, is an area of artificial life that models biological processes of development, in which there are layers of abstraction between a genotype and a phenotype. The phenotype begins with a seed or embryo and progresses toward maturity according to a set of rules or interactions. The individuals on which evolutionary or other forces are acting are these processes by which the embryo develops. Developmental systems and other forms of indirect encodings of solutions can be contrasted with direct encodings, in which each component of the phenotype is made explicit in the genotype. As the problems being solved by evolutionary computation have grown in complexity, scalability has become an important aspect of the design of new evolutionary computation techniques, and certain properties of developmental systems, such as compact genotypes, lend themselves well to scalability (Bentley and Kumar, 1999; Hornby and Pollack, 2001a), which motivated much early research on artificial development (Tufte, 2008). A system that uses artificial development may be called a generative or developmental system.

Often, these developmental encodings are based on biological developmental processes and principles. Doursat (2009) used lower-level developmental processes such as cell division/differentiation and morphogen gradients to create a self-patterning "organic canvas," and (Miller and Banzhaf, 2003) created a model for the programming of a cell, using cell division and simulated chemical environments, that was able to recreate a French flag and other patterns. Other approaches have involved the use of simulated gene regulatory networks (Guo et al., 2009), the exploitation of biological principles of degeneracy (Whitacre et al., 2010), and the evolution of grammars to generate programs or expressions in a given language (O'Neill and Ryan, 2001).

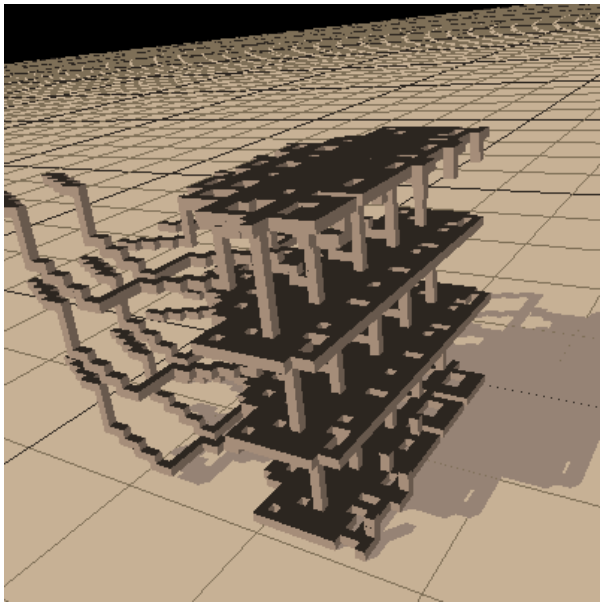

Figure 1: Visualization of an example brick "table" structure produced by GENRE.

Below, we provide a brief overview of the two different generative systems used in this study.

GENRE

GENRE (Hornby and Pollack, 2001b) is a developmental system that was designed to create more complex virtual creatures than had been created using earlier artificial life techniques. It took a grammatical approach, evolving Lindenmayer systems (L-systems), (Lindenmayer, 1968), parallel grammatical rewriting rules originally designed to model plant growth, that took in parameters and would generate creatures with hundreds of components. The L-systems were applied iteratively to rewrite strings of commands through which to construct creatures or other structures, such that complex strings were constructed from simple ones. Hornby analogized the parallel nature of the rules, and the repetitive structures that they tended to produce, to concurrent cell division. The system outperformed a non-generative system on performance, creature size, and natural look, when applied to the design of mobile robots and block-based "table" structures. Hornby et al. 2001 observed that the generated robots appeared to exhibit modular properties, but did not attempt to quantify this. An example GENRE-

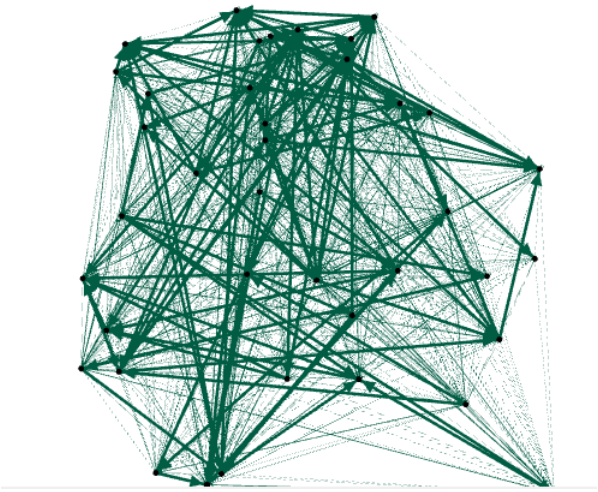

Figure 2: Visualization of an example artificial gene regulatory network (GRN) produced by GRNEAT.

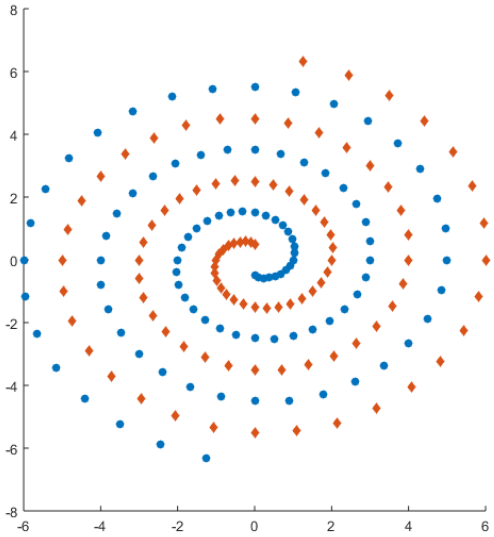

Figure 3: Illustration of two intertwining spirals, which must be distinguished from each other in the intertwining spirals problem.

produced structure can be seen in Fig.1.

GRNEAT

GRNEAT (Cussat-Blanc et al., 2015) evolves artificial gene regulatory networks, or GRNs (Banzhaf, 2003), which are simplified models of the genetic regulatory networks seen in biological systems, and used to control various kinds of agents. The GRNEAT algorithm evolves lists of proteins, which are then developed into network models with matrices of enhancing and inhibiting weights between nodes, mimicking the developmental module function of biological GRNs. The protein lists are used to initialize a GRN, which then updates its weights by calculating interactions between the proteins. It is based on NEAT (Stanley and Miikkulainen, 2002), a well-known algorithm for evolving neural networks, and retains NEAT's major distinguishing features: initialization with small networks, a crossover operator that preserves subnetworks during GRN recombination, and the use of speciation to give growth opportunity to potentially promising innovations. However, a key difference is that artificial GRNs are inherently a developmental encoding, as biological gene regulatory networks are, while NEAT is a direct encoding algorithm. An example GRNEAT network, visualized in Gephi (Bastian et al., 2009), can be seen in Fig.2

Methods

To test the effects of development on hierarchical modularity, we used the GENRE algorithm, which uses L-systems to model parallel cell division, and the GRNEAT algorithm, which evolves protein lists for construction of artificial gene regulatory networks in a manner mimicking NEAT's neuroevolution methods, both of which are described briefly above. We chose to compare GRNEAT to NEAT because, as stated above, GRNs are developmental by nature, so we could not simply compare a developmental GRN encoding to a nondevelopmental one. We ran GENRE on the bricktable-building problem that was one of its original test problems in (Hornby et al., 2001), which rewards individuals for minimizing the number of bricks and maximixing height, surface area, volume, and stability, and compared the results to those obtained by a non-developmental evolutionary algorithm that is packaged with GENRE for the purpose of running comparisons. We ran the GRNEAT evolutionary process on the problem of distinguishing two intertwined spirals, also called the intertwined spirals problem (Lang, 1988), which is illustrated in Fig.3, with fitness being measured by error ranging from -1 to 0, and compared the results to those obtained by running both feedforward and recurrent versions of the NEAT4J open source Java implementation of NEAT (Simmerson, 2006) on the same problem.

Key parameters for GENRE and for GRNEAT/NEAT are listed in Table 1 and Table 2 respectively. While the comparisons between GRNEAT and NEAT were done primarily using simulations of 250 generations, we also did a set of runs of GRNEAT that were only 10 generations, to see whether any modularity that existed was actually emerging over time or was present early in the simulation. We did not do 10-generation runs for NEAT because it had made almost no progress at solving the intertwining spirals problem after only 10 generations. In the NEAT4J implementation

of NEAT, there is an option to allow or disallow recurrent neural networks. We decided to allow recurrency for a more even comparison, as GRNEAT produces recurrent networks. To prevent either the GRNs or neural networks simply memorizing a sequence of outputs rather than learning a mapping from coordinates to spiral ID, in both GRNEAT and NEAT, we used a fresh copy of the pre-initialized network for each new input.

Problem Version	Trials	Generations	Num R, P, C
GENRE	10	100	10, 2 , 2
Nongenerative	10	100	1, NA, NA

Table 1: Key parameters in GENRE experiments. R is the number of production rules, P is the number of parameters per rule, C is the number of condition-successor pairs per rule.

Problem Version	Trials	Generations	pC, pM, PopSize
GRNEAT	20	10	0.25, 0.75, 500
GRNEAT	20	250	0.25, 0.75, 500
NEAT	20	250	0.25, 0.75, 500

Table 2: Key parameters in GRNEAT experiments. pC is probability of crossover, pM is probability of mutation, PopSize is Population Size.

In order to look at the quantitative modularity of GENRE's and its non-developmental counterpart's brick table structures, we needed to represent the structures as networks. In order to do that, we defined each brick as a node, and each case of a face of one brick touching a face of another brick as a link. The link structure was binary, with all links being represented in the adjacency matrix as having a value of 1, and all other elements of the adjacency matrix having a value of zero. This was not necessary for GRNEAT/NEAT, as both GRNs and neural networks are already represented as networks, with links having non-binary weights. Since GRNEAT produces both a matrix of enhancement weights and a matrix of inhibition weights, we combined them into a single weight matrix by subtracting the inhibition factors from the enhancement factors.

Many artificial life and theoretical biology studies of modularity use the metric Q, defined by the approach of (Newman and Girvan, 2004). This approach determines Q by looking at the percentage of edges in the network that connect nodes in the same module, and substracts the expected value for that percentage in a network with the same number of modules but random connections. The modules are defined by a previous part of the algorithm that splits the network into the modules that would maximize Q. Mathematically, the equation for Q in the Newman-Girvan algorithm is:

$$Q = \sum_{s=1}^{k} \left[\frac{l_s}{L} - \left(\frac{d_s}{2L} \right)^2 \right] \tag{1}$$

where L is the number of edges, K is the number of modules, d_s is the sum of degrees of nodes in module s, and l_s is the number of edges in that module.

This method is very useful for examining a single layer of modularity in binary networks (i.e. networks where there is either a connection between two nodes or there is not). However, the weights of links between nodes in GRNs can vary by several orders of magnitude. Both GRNs and recurrent neural networks may benefit from a modularity metric that can account for directedness. And the Newman-Girvan approach only looks for one layer of modularity, rather than for hierarchical modularity. Accordingly, we used the "Louvain method," which was designed for speed, maximization of community detection, and the detection of hierarchical levels of modularity, to determine Q (Blondel et al., 2008). In the Louvain method, each node in the network is initially assigned to its own module, and the modularity Q is calculated according to the following equation for a weighted graph:

$$Q = \frac{1}{2m} \sum ij \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta \left(c_i, c_j \right) \tag{2}$$

where A_{ij} represents edge weight between nodes i and j, m represents half the sum of the graph's edge weights, δ is a delta function, c_i and c_j are node communities, and k_i and k_j are the sums of the weights of all edges attached to node i and node j respectively.

Then, for each node, the algorithm calculates the change in modularity, the equation for which depends on whether the version of the algorithm for directed or undirected graphs is being used, for moving that node into the module of each of its neighbors. Once this change is calculated for all modules that the node is connected to, the node is moved into the module that would result in the greatest modularity increase (or left in place if no modularity increase is possible). If no increase is possible, the first level of Q is equal to the current modularity of the network. Subsequent, hierarchical levels of Q are calculated the same way in subequent phases of the algorithm, by using the modules from the previous level as nodes in a new network. For our study, we used Antoine Scherrer's MATLAB implementation of the Louvain method (Scherrer, 2008).

Because modularity can be positive or negative (where negative modularity means that there is less internal integration among modules than one would expect to see in a random graph), we defined a level of modularity as occurring when the Louvain algorithm produces a positive modularity

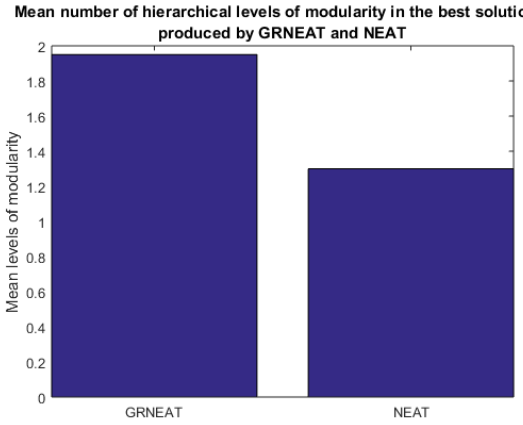

Figure 4: The mean number of levels of hierarchical modularity produced by a developmental network-evolving system, GRNEAT, was higher than that produced by the non-developmental one on which it is based, NEAT. The 95% CIs for GRNEAT and NEAT were 1.85-2.05 and 1.09-1.51. Number of trials N = 20, $p < 0.0001$

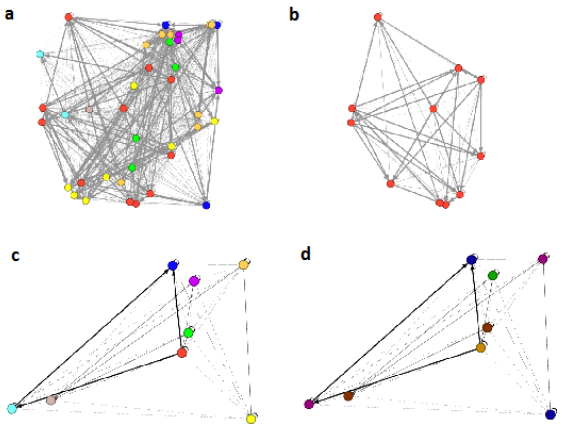

Figure 5: a) A GRNEAT-produced GRN with the links lightened for easier viewing. The nodes are colored according to the 8 first-level modules found by the first phase of the Louvain algorithm, which found that this level of modularity had $Q = 0.3018$. b) A single module of the GRN. c) The network of the next level of hierarchy, with each of its node representing, and color-coded as, a module from the previous level. d) The same next-level network, with $Q = 0.3266$, with the nodes colored in five new colors according to the 5 second-level modules found by the second phase of the Louvain algorithm.

value, with an allocation of nodes into modules that (if there are lower levels) involves combining some or all modules from the next-lowest level. To determine whether the numbers of hierarchical levels of Louvain modularity were equal in our sets of results, we used Welch's t-test, a variant of the traditional Student's t-test that is robust to non-normality in data and difference in variance between samples. We did not test for differences in the actual Q values of the lowest or other levels, as they were tangential to the question of hierarchy. In practice, Q values for specific levels were between 0.12 and 0.52 for both GRNEAT and NEAT (with most being between 0.2 and 0.4, indicating moderate amounts of single-level modularity), and between 0.3 and 0.72 for both GENRE and its direct encoding counterpart.

Results and Discussion

Our first comparisons were between a set of 20 trials of GRNEAT on the intertwined sprials problem and 20 trials of NEAT with recurrency allowed on the intertwined sprials problem, with mutation probability = 0.75 and crossover probability = 0.25, across 250 generations. In Fig.4, we can see that the best solutions produced in the GRNEAT trials had a mean number of levels of modularity of 1.95, while those produced in the NEAT trials had a mean number of levels of modularity of 1.3, a full 33% lower. This difference in the means was statisically significant ($p < 0.0001$). The emergence of multiple levels of modularity in one GRN is shown in Fig.5.

We wanted to examine whether the increased levels of modularity seen in GRNEAT were something that was emerging rather than something hard-coded into all GRNs. Because the average fitness of the NEAT neural networks after 250 generations was notably worse than that of the GRNEAT GRNs, and the networks notably smaller (see Fig.6), we also wanted to compare the NEAT networks to GRNs of more similar fitness and size. Accordingly, we compared the 20 250-generation GRNEAT trials to 20 10-generation trials (Fig.7, and, as the 10-generation GRNEAT GRNs were similar in size and fitness to the NEAT neural networks, we compared the 10-generation GRNEAT trials to the NEAT trials (Fig.8)

We can see from these figures that the mean hierarchical modularity of GRNEAT-produced GRNs (along with the size) has increased by nearly a third (a mean 1.5 levels of modularity vs 1.95 levels) between the 10th and 250th generations. We can also see tentative evidence (with a p value that is low but not statistically significant) that GRNEAT GRNs already have greater hierarchical modularity after 10 generations than NEAT recurrent neural networks have after 250, despite being nearly the same size, which is suggestive that this increased hierarchy is not solely a function of network size.

While the most obvious difference between GRNEAT and NEAT is the artificial development aspect, it is possible that there is some other factor influencing the development of hierarchical modularity. Therefore, we looked at a different developmental system using a very different

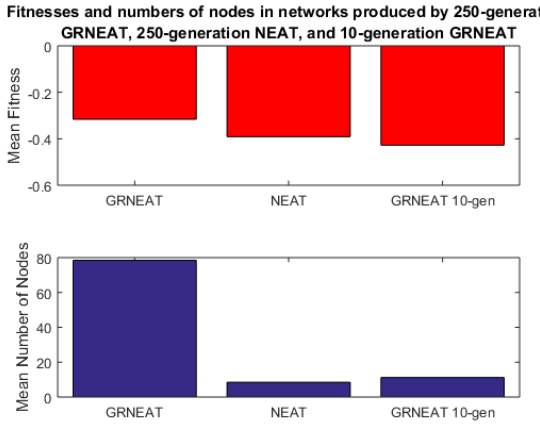

Figure 6: 250 generations of GRNEAT produced top solutions with greater mean fitness than 250 generations of NEAT. 250 generations of NEAT still had greater fitness than 10 generations of GRNEAT.

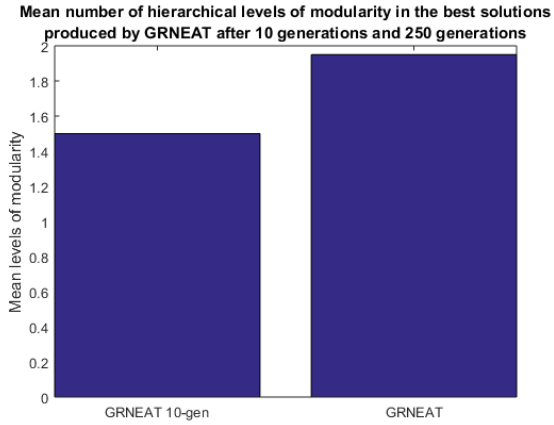

Figure 7: Networks produced by GRNEAT after 250 generations had greater mean levels of hierarchical modularity than networks produced by GRNEAT after 10 generations. The 95% CIs for GRNEAT and 10-generation GRNEAT were 1.85-2.05 and 1.28-1.72. Number of trials N = 20, p = 0.0014

mechanism of development, GENRE, and compared it to the direct encoding algorithm packaged with its implementation on the GENRE homepage (Hornby, 2001), as discussed in the Methods section. One useful aspect of looking at GENRE's brick-table structures is that the default fitness function for these structures encourages minimizing the number of blocks while maximizing other structural criteria. While, because of the nature of the block structures, the GENRE network representations were much larger than the GRNEAT or NEAT representations, they were actually smaller than those of the direct encoding alternative (640 blocks vs 768 blocks). Therefore, the possibility of network size being a major contributor to the different levels of hierarchy produced by a developmental vs a direct encoding is addressed.

As can be seen in Fig.9, there is a statistically significant (p = 0.0246) difference between the levels of hierarchical modularity produced by GENRE as compared to its nondevelopmental alternative, where the GENRE-produced structures had an average of 4.7 levels, and the others had an average of 4.2.

Notably, in both cases, the number of levels was far higher than for GRNEAT or NEAT, regardless of development, and the network sizes were much larger, which suggests that network size may play some role in the number of levels of hierarchical modularity. However, the fact that GENRE structures have more levels than do a nondevelopmental algorithm optimizing for the same fitness function, in the same number of generations, despite being 17% smaller, provides further evidence that the developmental encoding is doing some of the work in the emergence of this hierarchy.

Conclusion and Future Work

This work opens up two different areas of study. One is the study of the effects of developmental encodings on the emergence of modularity. This is an underexplored area, with the potential to contribute to our understading of the emergence of modularity in general. To our knowledge, this is the first time that developmental encoding effects on the emergence of modularity have been quantified. Another is the study of hierarchical modularity. While modularity is an active area of research, as we outlined earlier in this paper, the hierarchical aspect of modularity has been ignored in modularity-emergence studies despite the importance of hierarchy in biological and other understandings of modularity. This paper takes a first step toward remedying this oversight.

Our results suggest several more specific avenues for future work. It may be useful to compare other developmental systems to similar nondevelopmental ones, to see whether the same effect is observed. Even though GENRE and GRNEAT use very different mechanisms for encoding development, increasing the number of systems studied may provide more evidence that the effect seen here is mechanism-independent. Another possibility would be to adjust different parameters within the developmental systems, to see if other factors can be identified that promote the emergence of hierarchical modularity. Finally, it would be interesting to study the effects of development in biological systems, as was done in (Kashtan et al., 2007; Parter et al., 2007) to determine the effect of varying goals on modularity.

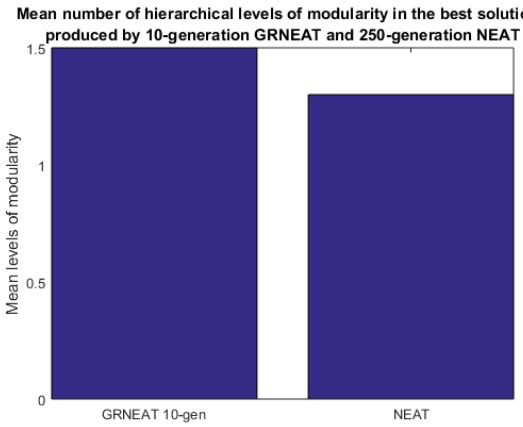

Figure 8: The mean number of levels of hierarchical modularity produced by only 10 generations of GRNEAT was higher than that produced by 250 generations of NEAT. The 95% CIs for 10-generation GRNEAT and NEAT were 1.28-1.72 and 1.09-1.51. Number of trials N = 20, p = 0.2

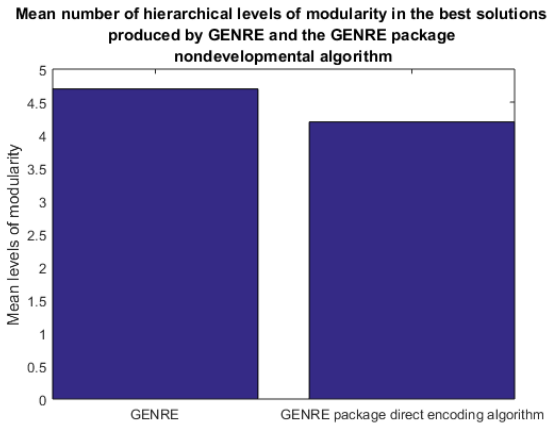

Figure 9: The mean number of levels of hierarchical modularity produced by a developmental brick-table-evolving algorithm, GENRE, was higher than that produced by the nondevelopmental algorithm with which it is packaged. The 95% CIs for GENRE and the nondevelopmental algorithm were 4.4-5.0 and 3.94-4.46. Number of trials N = 10, p = 0.0246

Acknowledgments.

Many thanks to Kyle Harrington for his help in troubleshooting GRNEAT and other code and his valuable feedback.

References

Banzhaf, W. (2003). On the dynamics of an artificial regulatory network. In *Advances in Artificial Life*, pages 217–227. Springer.

Bastian, M., Heymann, S., Jacomy, M., et al. (2009). Gephi: an open source software for exploring and manipulating networks. *ICWSM*, 8:361–362.

Bentley, P. J. and Kumar, S. (1999). Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *GECCO*, volume 99, pages 35–43.

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008.

Bolker, J. A. (2000). Modularity in development and why it matters to evo-devo. *American Zoologist*, 40(5):770–776.

Calcott, B. (2014). Chaining distinct tasks drives the evolution of modularity. In *ALIFE 14: The Fourteenth Conference on the Synthesis and Simulation of Living Systems*, volume 14, pages 701–702.

Clune, J., Mouret, J.-B., and Lipson, H. (2013). The evolutionary origins of modularity. *Proceedings of the Royal Society of London B: Biological Sciences*, 280(1755):20122863.

Cussat-Blanc, S., Harrington, K., and Pollack, J. (2015). Gene regulatory network evolution through augmenting topologies. *Evolutionary Computation, IEEE Transactions on*, 19(6):823–837.

Doursat, R. (2009). Organically grown architectures: Creating decentralized, autonomous systems by embryomorphic engineering. In *Organic Computing*, pages 167–199. Springer.

Guo, H., Meng, Y., and Jin, Y. (2009). A cellular mechanism for multi-robot construction via evolutionary multi-objective optimization of a gene regulatory network. *BioSystems*, 98(3):193–203.

Hartwell, L. H., Hopfield, J. J., Leibler, S., and Murray, A. W. (1999). From molecular to modular cell biology. *Nature*, 402:C47–C52.

Hornby, G. S. (2001). Genre homepage. online, 2001. `http://www.demo.cs.brandeis.edu/pr/evo_design/evo_design.html#genre_source`.

Hornby, G. S., Lipson, H., and Pollack, J. B. (2001). Evolution of generative design systems for modular physical robots. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 4, pages 4146–4151. IEEE.

Hornby, G. S. and Pollack, J. B. (2001a). The advantages of generative grammatical encodings for physical design. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 600–607. IEEE.

Hornby, G. S. and Pollack, J. B. (2001b). Evolving l-systems to generate virtual creatures. *Computers & Graphics*, 25(6):1041–1048.

Juille, H. and Pollack, J. B. (1996). Co-evolving intertwined spirals. In *in Proceedings of the Fifth Annual Conference on Evolutionary Programming*. Citeseer.

Kashtan, N. and Alon, U. (2005). Spontaneous evolution of modularity and network motifs. *Proceedings of the National Academy of Sciences of the United States of America*, 102(39):13773–13778.

Kashtan, N., Noor, E., and Alon, U. (2007). Varying environments can speed up evolution. *Proceedings of the National Academy of Sciences*, 104(34):13711–13716.

Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press.

Lang, K. J. (1988). Learning to tell two spirals apart. In *Proc. of 1988 Connectionist Models Summer School*.

Lindenmayer, A. (1968). Mathematical models for cellular interactions in development ii. simple and branching filaments with two-sided inputs. *Journal of Theoretical Biology*, 18(3):300–315.

Lipson, H., Pollack, J. B., Suh, N. P., and Wainwright, P. (2002). On the origin of modular variation. *Evolution*, 56(8):1549–1556.

Lowell, J. and Pollack, J. (2014). The effect of connection cost on modularity in evolved neural networks. In *ALIFE 14: The Fourteenth Conference on the Synthesis and Simulation of Living Systems*, volume 14, pages 726–733.

Miller, J. F. and Banzhaf, W. (2003). Evolving the program for a cell: from french flags to boolean circuits. *On Growth, Form and Computers*, pages 278–301.

Newman, M. and Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113.

O'Neill, M. and Ryan, C. (2001). Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358.

Parter, M., Kashtan, N., and Alon, U. (2007). Environmental variability and modularity of bacterial metabolic networks. *BMC evolutionary biology*, 7(1):169.

Scherrer, A. (2008). Matlab louvain implementation. online, 2008.

Schilling, M. A. (2002). Modularity in multiple disciplines. *Managing in the modular age: Architectures, networks and organizations*, pages 203–214.

Simmerson, M. (2006). Neat4j homepage. online, 2006.

Simon, H. A. (1996). *The sciences of the artificial*. MIT press.

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

Thompson, A. (2012). *Hardware Evolution: Automatic design of electronic circuits in reconfigurable hardware by artificial evolution*. Springer Science & Business Media.

Tufte, G. (2008). Phenotypic, developmental and computational resources: scaling in artificial development. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 859–866. ACM.

Vassilev, V. K., Job, D., and Miller, J. F. (2000). Towards the automatic design of more efficient digital circuits. In *Evolvable Hardware, 2000. Proceedings. The Second NASA/DoD Workshop on*, pages 151–160. IEEE.

Whitacre, J. M., Rohlfshagen, P., Bender, A., and Yao, X. (2010). The role of degenerate robustness in the evolvability of multiagent systems in dynamic environments. In *Parallel Problem Solving from Nature, PPSN XI*, pages 284–293. Springer.