Indirect Encoding of Neural Networks for Scalable Go

In: ***Proceedings of the 11th International Conference on Parallel Problem Solving From Nature (PPSN 2010)*. New York, NY: Springer**

Jason Gauci and Kenneth O. Stanley

School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816
jgauci@eecs.ucf.edu, kstanley@eecs.ucf.edu

Abstract. The game of Go has attracted much attention from the artificial intelligence community. A key feature of Go is that humans begin to learn on a small board, and then incrementally learn advanced strategies on larger boards. While some machine learning methods can also scale the board, they generally only focus on a subset of the board at one time. Neuroevolution algorithms particularly struggle with scalable Go because they are often directly encoded (i.e. a single gene maps to a single connection in the network). Thus this paper applies an *indirect encoding* to the problem of scalable Go that can evolve a solution to 5×5 Go and then *extrapolate* that solution to 7×7 Go and continue evolution. The scalable method is demonstrated to learn faster and ultimately discover better strategies than the same method trained on 7×7 Go directly from the start.

1 Introduction

The game of Go has proven challenging for artificial intelligence because the branching factor and state space in Go render traditional approaches intractable [1]. Go demands new search techniques to reduce the branching factor, and abstract representations that can consolidate the state space. One promising such approach is machine learning, wherein techniques such as temporal difference learning or neuroevolution learn a value function from an abstract representation [2–4].

Yet even with such innovations, experienced human Go players can still consistently defeat the strongest of computer players without a handicap [5]. One notable difference between human players and most machine learning-based approaches to Go is that the human player begins to learn Go on a small board [6]. Humans can then *extrapolate* information learned on the smaller board to a larger board, thereby bootstrapping from it. Such extrapolation is challenging for machine learning algorithms, which often cannot transfer knowledge from one board size to another.

However, several notable exceptions exist that typically fall into one of two categories: (1) The first convert the Go board into a set of local features that are independent of the board size [2]; (2) the second class of methods scan sections of the board and *remember* notable positions and information [3, 4]. In both cases, the key is to view a small section of the Go board at one time. As a result, it is potentially difficult to learn tactics (e.g. ladders) that depend on a holistic view of the board.

In this paper, a new method of scaling is presented that breaks from the aforementioned techniques, yet can still scale the board to new sizes and continue learning.

The method is based on Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT), which evolves artificial neural networks (ANNs) that are *aware of* and *parametrized by* the geometry of the board. As a result, these ANNs are able to make holistic decisions based on seeing the entire Go board at once. HyperNEAT encodes ANNs through an indirect representation that has the ability to scale the Go board to new sizes *without changing the representation* and continue evolution. The result is that candidates evolved on 5×5 Go and then scaled and evolved further at 7×7 Go outperform candidates evolved solely on 7×7 Go without scaling. Thus the main contribution is to show that indirect encoding is a viable foundation for training scalable learners, and offers the unique potential to represent holistic solutions at variable sizes.

2 Background

In Go, two players take turns placing stones on an $n \times n$ grid. The standard board size is 19×19; however, common board sizes also include 5×5 and 9×9. The objective is to possess more stones on the board than the opponent at the end of the game. If a player is able to form a complete border around a group of the opponent's stones, the surrounded stones are removed from the board. The player with the most stones at the end is declared the winner. A complete description of Go can be found in Botermans [5] and Shotwell [6].

Go is designed for play at several board sizes. However, few machine learning methods can modify the board size in the middle of training and continue learning. This section discusses several exceptions and reviews the NEAT and HyperNEAT methods.

2.1 Reinforcement Learning and Scalable Go

Because the strategies for 19×19 boards are very different than those for e.g. 9×9, players transitioning from small to large boards must continue to learn and refine their strategy and tactics [6]. Ideally, machine learning algorithms should also learn to play Go at varying board sizes without discarding tactics learned on smaller boards and starting from scratch.

Reinforcement learning has been applied to scalable Go through several approaches [2–4]. Silver et al. [2] introduce the idea of assigning a weight to each shape in a *shape set*. The key idea is that all shapes learned on a smaller board are analogous on a larger one. New shapes that exist only at the higher scale are introduced after scaling by initializing them with a weight of 0. Silver et al. [7], Enzenberger [8], and Schraudolph et al. [9] follow a similar approach.

In a different approach, Stanley and Miikkulainen [4] evolved a neural network that controls a robot eye that has a small field of vision. The robot is able to move across the board and place pieces. Because the field of vision for the robot is smaller than the size of the Go board, the robot can learn local concepts independently of location. As a result, the roving eye can learn to play Go at any resolution.

Schaul and Schmidhuber [3] introduced a neuroevolution-based action-value approximator for Go that evolves a Multi-Dimensional Recurrent Neural Network (MDRNN) [10]. The MDRNN performs swipes across the Go board. To perform a swipe, the same neural network is evaluated at every position of the Go board. In this

way, information is carried across the board through the output values. MDRNNs are inherently scalable because the network is only concerned with relative information.

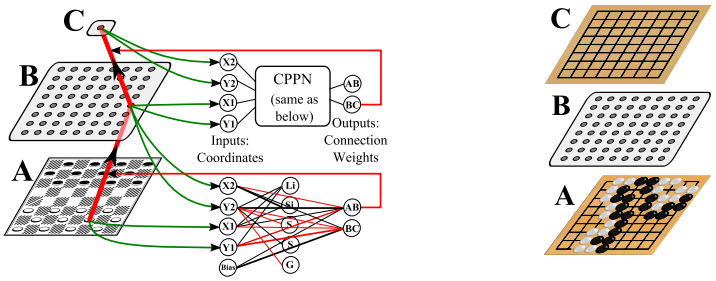
While these methods have learned effective Go players, each of them relies on integrating a set of small, local views that are processed independently over time or space. The danger is that less holistic heuristics that are significantly simpler become attractive local optima. In general, an interesting question is whether it is possible to scale the Go board to new resolutions while also processing the entire Go board without relying on subsquares. HyperNEAT, reviewed next, creates such a capability.

2.2 Indirect Encodings and HyperNEAT

The first methods to evolve both network structure and connection weights encoded networks *directly*, which means that a single gene in the genotype maps to a single connection in the phenotype [11]. NeuroEvolution of Augmenting Topologies (NEAT) is one such method [12]. In addition to evolving weights of connections, NEAT can build structure and add complexity. NEAT is a leading neuroevolution approach that has shown promise in board games and other challenging control and decision making tasks [4, 12, 13]. While this approach is straightforward, it requires learning each connection weight individually. Human engineering is one approach to overcoming this limitation. For example, Fogel [14] applies ANNs to checkers by dividing the board into subsquares and architecting the ANN to process them at different resolutions. However, ideally, evolution would capture patterns and regularities on its own.

Indirect encodings give evolution the opportunity to explore patterns and regularities by encoding the genotype as a *description* that maps indirectly to the phenotype [15–19]. That way, the genotype can be much smaller than the phenotype, which results in fewer variables to optimize for the evolutionary algorithm. *Compositional pattern producing networks* (CPPNs) are one such indirect encoding that draws inspiration from biology [20]. The idea behind CPPNs is that patterns such as those seen in nature can be described at a high level as a composition of functions that are chosen to represent several common motifs in patterns. The appeal of this encoding is that it allows patterns with regularities such as symmetry (e.g. with Gaussians), repetition (e.g. with periodic functions such as sine), and repetition with variation (e.g. by summing periodic and aperiodic functions) to be represented as networks of simple functions, which means that NEAT can evolve CPPNs just as it evolves ANNs.

Hypercube-based NEAT (HyperNEAT) is an algorithm that extends CPPNs, which encode two-dimensional spatial patterns, to also represent connectivity patterns [15, 21–25]. That way, NEAT can evolve CPPNs that encode ANNs with symmetries and regularities that are computed directly from the geometry of the task inputs. The key insight is that $2n$-dimensional spatial patterns are isomorphic to connectivity patterns in n dimensions, i.e. in which the coordinate of each endpoint is specified by n parameters. To apply HyperNEAT to checkers, for example, the substrate (which is the name for the set of ANN nodes and their geometry in HyperNEAT) input layer is arranged in two dimensions to match the geometry of the checkers board (figure 1a). To compute the weight of a connection, the CPPN encoding works by inputting the coordinates of its endpoints (i.e. x_1, y_1, x_2, and y_2) and outputting the connection weight. All connections are computed in this way, in effect painting a pattern across the network connectivity.



(a) Checkers Evaluation Function

(b) Go Action Selector

Fig. 1. Substrates for Board Games. Substrate (a) contains a two-dimensional input layer labelled *A* that corresponds to the geometry of a game board, an analogous two-dimensional hidden layer *B*, and a single-node output layer *C* that returns a board evaluation. The two CPPNs to the right of the board are depictions of the *same* CPPN being queried to determine the weights of two different substrate connections. In this way, a four-input CPPN can specify the connection weights of a two-layer network structure as a function of the positions, and hence the geometry, of each node. An action selector substrate (utilized in this paper) with an output for every possible move is shown in (b).

Gauci and Stanley [21, 25] originally introduced the type of representation in figure 1a for applying HyperNEAT to the game of Checkers. To distinguish the flow of information through the policy network from the geometry of the game, a third dimension in the substrate represents information flow from one layer to the next. Along this third dimension, the two-dimensional input layer connects to an analogous two-dimensional hidden layer so that the hidden layer can learn to process localized geometric features. The hidden layer then connects to a single output node, whose role is to evaluate board positions. The CPPN distinguishes the set of connections between the inputs and the hidden layer from those between the hidden layer and the output node by querying the weights of each set of connections from a separate output on the CPPN (note the two outputs in the CPPN depiction in figure 1a). That way, the x and y positions of each node are sufficient to identify the queried connection and the outputs differentiate one connection layer from the next. Because the CPPN can effectively compute connection weights as a function of the *difference* in positions of two nodes, it can easily map a repeating concept across the whole board.

This approach allows HyperNEAT to discover geometric regularities on the board by expressing connection weights as a function of geometry. For a full description of HyperNEAT see Stanley et al. [15] or Gauci and Stanley [25].

3 Approach: HyperNEAT in Go

Because of the large branching factor in Go [1], board evaluation functions such as the HyperNEAT approach to checkers discussed above may not be tractable in practice. In the case of Go, there can be hundreds of boards to evaluate in a single move, even at the lowest ply. Thus an appealing alternative would be an *action selector* that evaluates the current state and suggests where to move, rather than a board evaluation function that

must view many boards in the future to decide on a move. The next section explores this idea in more detail.

3.1 Evolving an Action Selector

Because HyperNEAT can evolve high-dimensional structure as an indirect encoding, it opens up the possibility to evolve an action selector. This type of ANN contains an output for *each possible action* (figure 1b). In this case, an output exists for each square on the Go board. By activating the substrate, HyperNEAT populates each output with a value indicating the desirability of putting a piece in that position on the Go board. Thus no forward search through the game tree is needed, thereby saving significant computation. Once the substrate has been activated, the output with the highest activation is chosen and the corresponding square on the Go board undergoes a sanity check that prevents the network from making invalid moves in the game. As a result of this new architecture, the output, hidden, and input layers of the Go substrate all contain $n \times n$ nodes, where n denotes the size of the board. Given a board size of 7×7, the substrate thus contains 147 nodes and 4,802 connections. Indirect encoding can produce the smooth patterns of weights necessary to begin evolution with so many connections and still learn effectively. The next section explores the *substrate extrapolation* method that allows solutions to scale in this paper.

3.2 Substrate Extrapolation

A major problem for traditional neuroevolution is that the number of evaluations to solve a problem is related to the number of connections in the network being evolved [12]. Training a network with ten million connections can require significantly more evaluations than training one with one hundred. However, Stanley et al. [15] showed that it is possible to query the *same* CPPN at varying substrate resolutions to create larger ANNs. Thus a promising potential approach to expanding the action selector size is to learn basic concepts on a small substrate, increase the substrate resolution, and then continue learning more advanced concepts at the higher resolution. This approach is designed to allow early, rapid learning of fundamental concepts.

There are two ways in HyperNEAT to scale a substrate input layer that represents a geometric space. The first is to sample the inputs at a higher resolution. This form of scaling, called *continuous substrate extrapolation*, preserves the geometric relationships between locations on the input signal (figure 2a). The two images, while different resolutions, exist within the same geometric area. That is, a specific location in the image does not change its *meaning* even if the resolution of the image changes. Thus the scaling changes *only* the distance between two adjacent pixels. Because CPPN inputs are by convention limited to a domain of $[-1,1]$, the CPPN effectively normalizes the width and height of the image regardless of resolution, and can thereby extrapolate the ANN to handle this form of scaling naturally. Stanley et al. [15] demonstrated such continuous substrate extrapolation with HyperNEAT in a simple visual recognition domain.

While this method can be effective in visual tasks, some domains do not lend themselves to this form of scaling. For example, if the resolution of the Go board in figure 2b

(a) Continuous Extrapolation (b) Discrete Extrapolation

Fig. 2. Continuous Versus Discrete Extrapolation. In continuous substrate extrapolation (a), the bounds of the geometry do not change as the scale increases. In discrete extrapolation (b), the relative area of a single square stays the same, but the overall geometry is expanded outward. In this case, special care is needed to ensure that the network scales appropriately with the domain.

is increased, the size of the domain *itself* increases, as opposed to in the prior example, wherein it simply becomes more detailed. In such *discrete substrate extrapolation*, the size of a meaningful unit of information does not change as the resolution increases. As a result, a new method must be designed to handle this form of scaling.

3.3 Discrete Substrate Extrapolation Implementation

The problem in discrete extrapolation is that the range of the input domain changes as the scale increases. To address this phenomenon, it is necessary to first decide on the maximum resolution of the system. In Go, this maximum resolution is 19×19, the size of the largest tournament Go board. The next step is to calculate the distance between two adjacent cells at this resolution. Because each input to the CPPN ranges from -1 to 1, the Go board must be rescaled to fit this new range. Thus the Go board position at index 0 maps to -1 and the position at index 18 maps to 1, and the distance between two adjacent cells in the Go board is therefore $\frac{2}{18}$. Interestingly, if the system is trained first at a lower resolution, e.g. 5×5, the smaller domain can be situated in the very same coordinate system (figure 2b). Increasing the resolution of each substrate layer during evolution is then an effective method to allow holistic complexification.

4 Experiment

The experiment in this paper aims to determine the effects of scaling HyperNEAT substrates on evolved Go action selectors. The player begins by playing ten games of Go against a fixed policy on a 5×5 board for 500 generations. The fixed policy player is *Liberty Player* from the SimplePlayers package of Fuego [26], who "tries to capture and escape with low liberty stones." A *liberty stone* is surrounded on three of the four sides with stones, and only has one empty adjacent space (i.e. one liberty). Liberty Player can be applied to boards of any size. Because Liberty Player places stones adjacent to stones with few liberties, it escapes captures and also quickly captures given the opportunity. When two or more potential moves are equally viable, Liberty Player picks one at random. These factors make Liberty Player a nontrivial opponent that provides sufficient challenge to demonstrate the utility of scaling. After training on a 5×5 Go board,

the domain switches to playing Go against the same policy on a 7×7 board. Like the evolved player, Liberty Player is an action selector, that is, it only evaluates the current board and returns a location on which to place a stone.

During evolution, each candidate plays ten games of Go against the Liberty Player. After each game has ended, the candidate receives a reward based on the final score and the size of the board.

$$\text{fitness} = \begin{cases} 8b^2 & \text{if the evolved player wins} \\ \max\left(0, s + 2b^2\right) & \text{if the evolved player loses,} \end{cases} \tag{1}$$

where s denotes the final score and b denotes the size (i.e. length) of the board. This fitness function guarantees that all individuals will receive a positive fitness (as HyperNEAT requires), and that negative Go scores will still result in a positive reward. This convention puts additional emphasis on winning and also avoids rewarding individuals who win by a large margin in a single game, but lose the remaining games.

4.1 Experimental Parameters

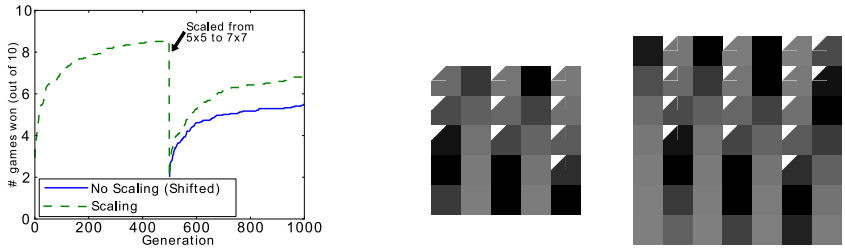
Parameter settings in the experiment follow precedent in applying HyperNEAT to checkers [21, 25]. The population size was 100 and each run lasted 500 generations. The disjoint and excess node coefficients were both 2.0 and the weight difference coefficient was 1.0. The compatibility threshold was 6.0 and the compatibility modifier was 0.3. The target number of species was eight and the drop-off age was 15. The survival threshold within a species was 20%. Offspring had a 3% chance of adding a node and a 5% chance of adding a link, and every link of a new offspring had an 80% chance of being mutated. Available CPPN activation functions were sigmoid, Gaussian, sine, and linear functions. Recurrent connections within the CPPN were not enabled. Signed activation was used in the CPPN and substrate, resulting in a node output range of $[-1, 1]$. By convention, a connection is not expressed if the magnitude of its weight is below a minimal threshold of 0.2 [22]; otherwise, it is scaled proportionally to the CPPN output. These parameters were found to be robust to variation in preliminary experimentation.

5 Results

To determine the effect of scaling, substrate extrapolation is compared to an unscaled approach that plays only 7×7 Go. Although fitness drives evolution, fitness cannot be a benchmark for scaling performance because it is derived from the Go score, which varies with the size of the board. Therefore, the win rate is recorded during evolution and determines the effective skill of the player for the purpose of comparing the scaled to non-scaled methods.

Figure 3a compares the performance of the non-scaled 7×7 method against the scaled substrate, averaged over 25 runs. Note that the non-scaled results are shifted to the right so that the reader can easily compare the effects of scaling to not scaling. The scaling approach won significantly more games than the non-scaling approach in all generations after 524 (i.e. 24 generations after scaling) ($p < 0.05$).

To give an idea how scaling works, figure 3b shows a single *receptive field* connecting to the center output from the hidden layer of a scalable substrate at the two



(a) Scaled versus non-scaled performance

(b) Receptive field at 5×5 and 7×7 scales

Fig. 3. Scaling Comparison and Visualization. The average performance of the generation champions over 25 runs of each variant is shown in (a). The performance is measured as the number of games won out of a possible 10 against Liberty Player. The scaled method wins significantly more than the non-scaled method in every generation beyond 524. A receptive field for the center output node on the substrate is shown in (b). Note that when the substrate is scaled to 7×7, the pattern is extrapolated outwards.

resolutions. Each grayscale box represents a link weight from a node in the hidden layer at that location to the center node of the output layer. White triangles in the corner of an box denote negative weights. The individual from which this receptive map was extracted is from generation 500, at which the domain is scaled to 7×7. Note that the pattern of weights is extrapolated outward as the substrate is scaled from 5×5 to 7×7. To understand this result, recall that the substrate is scaled with the discrete substrate extrapolation method. As a result, when the substrate is created at 5×5, the CPPN is queried with all possible combinations of the numbers $-\frac{2}{3}, -\frac{1}{3}, -0, \frac{1}{3}, \frac{2}{3}$ as inputs x_1, x_2, y_1, y_2. The choice of inputs to the CPPN explicitly defines the particular connection weight that the CPPN will output. The substrate is scaled to 7×7 by expanding the inputs to include all possible combinations of the numbers -1, $-\frac{2}{3}$, $-\frac{1}{3}$, -0, $\frac{1}{3}$, $\frac{2}{3}$, 1. This expansion adds the additional cells shown in 3b. This new pattern is thereby an effective bootstrap for learning more advanced concepts at the higher scale.

6 Discussion & Future Work

The key contribution of this paper is to show that indirect encoding makes possible a new kind of holistic, scalable Go player. Interestingly, an evaluation at 7×7 takes *ten times longer* than the same evaluation at 5×5 because the network size is larger and the games take more turns to complete. A method that can learn fundamental concepts at a low board size can thus more quickly progress to more advanced concepts at higher sizes, and thereby learn them with less computational overhead.

The CPPN encoding allows the HyperNEAT substrate to input and output an entire board of neurons. This method thus differs from other scalable approaches that either divide the board into local segments [3] or local features [2]. Constructing a function from the holistic board geometry is important for several reasons. First, it removes the need for a human or external process to divide the search space into local features or

segments. Second, constructing functions directly from geometry allow long-distance geometric relationships to be taken into account. For example, the decision to place a piece in Go often hinges not only on the position in the local area, but also on the state of conflicts elsewhere on the board and the geometric relationship of those conflicts with the local positions.

Future work will focus on incrementing to higher board sizes, evolving general Go players with HyperNEAT, and comparing them to other Go players. In addition, it is possible to bootstrap a Monte Carlo Tree Search (MCTS) algorithm with an action-evaluation function evolved by HyperNEAT. For example, the Upper Confidence Bounds Applied to Trees (UCT) algorithm is enhanced by adding a default policy [27]; however, the authors note that, "in many domains it is difficult to construct a good default policy." It is possible that HyperNEAT can evolve an effective default policy for UCT or any search algorithm.

7 Conclusion

This paper focused on the effects of scaling and demonstrated that players evolved incrementally through a scalable representation learn faster and more effectively than players evolved solely at the large scale. This result implies that fundamental concepts learned at a lower resolution facilitated further learning at the higher scale. The substrate extrapolation method scaled the information learned on the 5×5 Go board to the 7×7 board and the HyperNEAT algorithm was able to continue evolution at this new resolution. The main contribution is a step towards holistic neural strategies through indirect encoding that can be scaled to higher resolution or size.

References

[1] J. Burmeister and J. Wiles. The challenge of Go as a domain for AI research: A comparison between go and chess. In *Proceedings of the Third Australian and New Zealand Conference on Intelligent Information Systems. IEEE Western Australia Section*, pages 181–186, 1995.

[2] D. Silver, R. Sutton, and M. Müller. Reinforcement learning of local shape in the game of go. In *20th International Joint Conference on Artificial Intelligence*, pages 1053–1058, 2007.

[3] T. Schaul and J. Schmidhuber. Scalable neural networks for board games. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*. Springer, 2008.

[4] Kenneth O. Stanley and Risto Miikkulainen. Evolving a roving eye for Go. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, Berlin, 2004. Springer Verlag.

[5] Jack Botermans. *The Book of Games: Strategy, Tactics, and History*. Sterling Publishing Co., 2008.

[6] Peter Shotwell. *Go! More Than a Game.* Turtle Publishing, 2003.

[7] D. Silver, R.S. Sutton, and M. Müller. Sample-based learning and search with permanent and transient memories. In *Proceedings of the 25th international conference on Machine learning*, pages 968–975. ACM, 2008.

[8] M. Enzenberger. Evaluation in Go by a neural network using soft segmentation. In *Advances in computer games: many games, many challenges: proceedings of the ICGA/IFIP SG16 10th Advances in Computer Games Conference (ACG 10), November 24-27, 2003, Graz, Styria, Austria*, page 97. Kluwer Academic Pub, 2003.

[9] N.N. Schraudolph, P. Dayan, and T.J. Sejnowski. Temporal difference learning of position evaluation in the game of Go. *Advances in Neural Information Processing Systems*, pages 817–817, 1994.

[10] A. Graves, S. Fernandez, and J. Schmidhuber. Multi-dimensional recurrent neural networks. *Lecture Notes in Computer Science*, 4668:549, 2007.

[11] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[12] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.

[13] Kenneth O. Stanley and Risto Miikkulainen. Continual coevolution through complexification. In *Genetic and Evolutionary Computation Conference*, 2002.

[14] David B. Fogel. *Blondie24: Playing at the Edge of AI*. 2002.

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

[16] Petet J. Bentley and S. Kumar. 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)*, pages 35–43, 1999.

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

[18] Josh C. Bongard. Evolving modular genetic regulatory networks. In *Proceedings of the 2002 Congress on Evolutionary Computation*, 2002.

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

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

[21] Jason Gauci and Kenneth O. Stanley. 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, 2008. AAAI Press.

[22] Jason Gauci and Kenneth O. Stanley. Generating large-scale neural networks through discovering geometric regularities. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007)*, New York, NY, 2007. ACM Press.

[23] David D'Ambrosio and Kenneth O. Stanley. A novel generative encoding for exploiting neural network sensor and output geometry. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007)*, New York, NY, 2007. ACM Press.

[24] David B. D'Ambrosio and Kenneth O. Stanley. Generative encoding for multiagent learning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008)*, New York, NY, 2008. ACM Press.

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

[26] M. Enzenberger and M. Müller. Fuego–an open-source framework for board games and go engine based on monte-carlo tree search. Technical report, Technical Report TR09-08, University of Alberta, Edmonton, 2009.

[27] Sylvain Gelly and David Silver. Combining online and offline knowledge in uct. In *In Zoubin Ghahramani, editor, Proceedings of the International Conference of Machine Learning (ICML 2007)*, pages 273–280, 2007.