Generative Encoding for Multiagent Learning

David B. D'Ambrosio
Evolutionary Complexity Group
School of EE and CS
University of Central Florida
Orlando, FL 32816, USA
ddambro@eecs.ucf.edu

Kenneth O. Stanley
Evolutionary Complexity Group
School of EE and CS
University of Central Florida
Orlando, FL 32816, USA
kstanley@eecs.ucf.edu

ABSTRACT

This paper argues that multiagent learning is a potential "killer application" for generative and developmental systems (GDS) because key challenges in learning to coordinate a team of agents are naturally addressed through indirect encodings and information reuse. For example, a significant problem for multiagent learning is that policies learned separately for different agent roles may nevertheless need to share a basic skill set, forcing the learning algorithm to reinvent the wheel for each agent. GDS is a good match for this kind of problem because it specializes in ways to encode patterns of related yet varying motifs. In this paper, to establish the promise of this capability, the Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) generative approach to evolving neurocontrollers learns a set of coordinated policies encoded by a *single* genome representing a team of predator agents that work together to capture prey. Experimental results show that it is not only possible, but *beneficial* to encode a heterogeneous team of agents with an indirect encoding. The main contribution is thus to open up a significant new application domain for GDS.

Categories and Subject Descriptors: I.2.6 [**Artificial Intelligence**]: Learning—*connectionism and neural nets, concept learning*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*

General Terms: Algorithms

Keywords: CPPNs, HyperNEAT, Multiagent systems, NEAT, Neural Networks

1. INTRODUCTION

Presently, a major goal for generative and developmental systems (GDS) [1, 7, 12, 15, 19, 24] is to find a "killer application" that leverages the capability to efficiently encode complex patterns, which would help to attract attention outside the GDS community. At the same time, an important goal for broader artificial intelligence is to coordinate a team of agents that cooperate to complete a task [16, 26]. Machine learning is an appealing approach to constructing such *multiagent systems* because the best cooperative team policy may not be known a priori. While several approaches to learning have been applied to multiagent systems [11, 27], this paper argues that multiagent learning may be a "killer application" for GDS.

In fact, GDS is a natural fit for multiagent systems because it fundamentally exploits patterns within solutions, and patterns are precisely what characterize the makeup of multiagent teams. That is, the policies of agents on a team are often conceptually distributed in a spatial pattern according to their positions. For example, in a soccer (i.e. football) team, the positions closest to the goal are defensive and become incrementally more offensive the farther they are from the goal. Also importantly, even as policies vary across space, agents tend to *share* common skills; in the soccer example, all players know how to pass and kick. In fact, a significant problem for multiagent systems is that learning separate policies for separate agents can lead to reinventing the wheel many times when a core set of skills is needed by every agent. Variation on a policy theme distributed across space is thus reminiscent of the regular spatial patterns for which GDS is known [15, 20, 24], suggesting that GDS may be a promising new approach to multiagent learning.

To implement this idea in practice, Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT), which has demonstrated success as a GDS method that evolves connectivity patterns for large-scale artificial neural networks (ANNs) [5, 22], is extended to encode *patterns of ANNs distributed across space* from a single genome. The spatial distribution of ANNs matches with the locations of agents on the team, thereby allowing HyperNEAT to learn a *pattern of policies*, all generated from the same genome. In this way, these policies can naturally become variations on a theme.

To demonstrate its promise, the Multiagent HyperNEAT method is tested on several variants of a multiagent predator-prey problem in which a team of multiple predators is evolved to round up a team of prey that try to run away. This task is challenging because the predators must coordinate their behavior to avoid pushing the prey away from each other. To show that Multiagent HyperNEAT indeed gains an advantage by distributing heterogeneous policies across space, it is compared to evolved homogeneous teams, wherein a single policy is evolved by HyperNEAT that is assigned to every agent on the team. The result is that the heterogeneous teams evolved by Multiagent HyperNEAT significantly outperform homogeneous teams and exhibit highly coordinated multiagent tactics, all derived from a single genome. Fur-

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.

thermore, multiagent teams can be bootstrapped from a single seed policy, allowing them to immediately elaborate on a core set of skills, thereby improving performance even further.

Thus the main conclusion is that GDS offers an entirely new approach to multiagent learning that expands the available options to practitioners. By opening up a popular area of artificial intelligence to GDS, this paper expands the potential audience for future research in GDS.

2. BACKGROUND

This section reviews relevant multiagent approaches and the NEAT and HyperNEAT methods that form the backbone of Multiagent HyperNEAT.

2.1 Cooperative Multiagent Learning

Multiagent systems confront a broad range of domains, from predator-prey scenarios (as in this paper) to military tactics, creating the opportunity for real-world applications. In cooperative multiagent learning, which is reviewed in this section, agents are trained to work together, usually by one of several alternative methods.

Multiagent reinforcement learning (RL) is a popular approach that focuses on identifying and rewarding promising cooperative states and actions among a team of agents [11, 16]. However, multiagent domains are challenging because agents often do not have complete information about the rest of the team and their opponents. Even in scenarios with complete state information, the number of states grows exponentially with the number of agents, making the state space expensive to represent.

An alternative approach, cooperative coevolution, is an established evolutionary method for training teams of agents that must work together [16, 17, 18]. In this approach fitness is assigned to agents based on their ability to perform a task with other evolving agents. Coevolution is attractive because the agents are explicitly evaluated based on their teamwork as well as their ability to complete a task. However, a downside is that, depending on the population model chosen, the agents may either easily specialize yet not share skills, or may share skills yet specialize poorly [27].

An entirely different approach to training separate agents, which assumes global communication, is a single, monolithic genome whose phenotype controls all the agents simultaneously. Such consolidation allows information sharing but generally increases the dimensionality of the search and ignores separability [28]. A related approach is to directly encode several disconnected policies in a single monolithic genome [2, 13], which gains separability at the expense of information sharing. One solution to reduce dimensionality in either case without combining multiple individuals is to assign the same homogeneous control system to each agent [3]. If all the agents are controlled by separate instantiations of a single controller, then it is only necessary to discover that one policy, and the problem of sharing discoveries disappears. However, Yong and Miikkulainen [28] show that in predator-prey tasks with three predators and one prey, heterogeneous teams learn more effective strategies than homogeneous ones.

Thus the challenge in multiagent learning is to devise an approach that balances the need to reduce dimensionality with the desire for heterogeneity and the need for shared skills. The next section reviews the NEAT method, the foundation for the multiagent learning approach introduced in this paper, which aims to strike such a balance.

2.2 NEAT

NeuroEvolution of Augmenting Topologies (NEAT) evolves ANNs that perform well in a variety of control and decision-making problems [21, 23, 25]. It starts with a population of small, simple neural networks and then *complexifies* them over generations by adding new nodes and connections through mutation. By evolving networks in this way, the topology of the network does not need to be known a priori; NEAT searches through increasingly complex networks to find a suitable level of complexity.

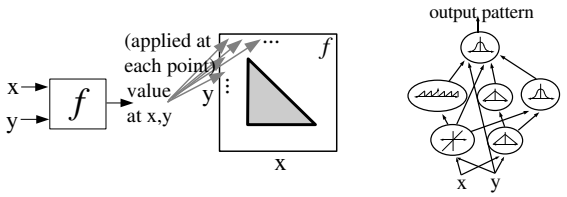
To keep track of which gene is which while new genes are added, a *historical marking* is uniquely assigned to each new structural component. During crossover, genes with the same historical markings are aligned, producing meaningful offspring efficiently. Speciation in NEAT protects new structural innovations by reducing competition between differing structures and network complexities, thereby giving newer, more complex structures room to adjust. Networks are assigned to species based on the extent to which they share historical markings. Complexification, which resembles how genes are added over the course of natural evolution [14], is thus supported by both historical markings and speciation, allowing NEAT to establish high-level features early in evolution and then later elaborate on them. Stanley and Miikkulainen [23, 25] provide a complete overview of the NEAT method. The next section explains how NEAT is extended to a developmental system with an indirect encoding.

2.3 CPPNs and HyperNEAT

The standard NEAT algorithm is not a generative or developmental system because its encoding is *direct*, that is, each gene maps to a single piece of structure in the phenotype. In contrast, GDS relies on *indirect* encodings because they allow genetic information to be reused so that the phenotype potentially contains more components than the genotype contains genes. Indirect encodings are often motivated by *development* in biology, in which the genotype maps to the phenotype indirectly through a process of growth [1, 7, 12, 15, 19, 20, 24].

Recently, NEAT has been extended to evolve a high-level developmental abstraction called Compositional Pattern Producing Networks (CPPNs) [20]. The idea behind CPPNs is that patterns in nature can be described at a high level as *compositions of functions*, wherein each function in the composition represents a stage in development. For example, a Gaussian function can represent a symmetric chemical gradient forming over time. Each component function also creates a novel geometric *coordinate frame* within which other functions can reside. The appeal of this encoding is that it allows developmental processes to be represented as networks of simple functions, which means that NEAT can evolve CPPNs just like ANNs. CPPNs are similar to ANNs, but they rely on more than one activation function (each representing a chemical gradient common to development) and are an abstraction of a different biological process. Also, unlike most generative and developmental encodings, CPPNs do not require an explicit simulation of growth or local interaction, yet still exhibit their essential capabilities [20].

Specifically, CPPNs produce a phenotype that is a function of n dimensions, where n is the number of dimensions



(a) Mapping (b) Composition

Figure 1: CPPN Encoding. (a) The function f takes arguments x and y, which are coordinates in a two-dimensional space. When all the coordinates are drawn with an intensity corresponding to the output of f, the result is a spatial pattern, which can be viewed as a phenotype whose genotype is f. (b) The CPPN is a graph that determines which functions are connected. The connections are weighted such that the output of a function is multiplied by the weight of its outgoing connection.

in physical space. For each coordinate in that space, its level of expression is an output of the function that encodes the phenotype. Figure 1 shows how a two-dimensional phenotype can be generated by a function of two parameters that is represented by a network of composed functions. Because CPPNs are a superset of traditional ANNs, which can approximate any function [4], CPPNs are also universal function approximators. Thus a CPPN can encode any pattern within its n-dimensional space.

The main idea in HyperNEAT is to extend CPPNs, which encode spatial patterns, to also represent *connectivity patterns* [5, 8, 22]. That way, NEAT can evolve CPPNs that represent large-scale ANNs with their own symmetries and regularities. 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.

Consider a CPPN that takes four inputs labeled x_1, y_1, x_2, and y_2; this point in four-dimensional space *also* denotes the connection between the two-dimensional points (x_1, y_1) and (x_2, y_2), and the output of the CPPN for that input thereby represents the weight of that connection (figure 2). By querying every possible connection among a set of points in this manner, a CPPN can produce a neural network, wherein each queried point is a neuron position. Because the connections are produced by a function of their endpoints, the final structure is produced with *knowledge* of its geometry. In effect, the CPPN is painting a pattern on the inside of a four-dimensional hypercube that is interpreted as an isomorphic connectivity pattern, which explains the origin of the name *Hypercube-based NEAT* (HyperNEAT). Connectivity patterns produced by a CPPN in this way are called *substrates* so that they can be verbally distinguished from the CPPN itself, which has its own internal topology.

Each queried point in the substrate is a node in a neural network. The experimenter defines both the location and role (i.e. hidden, input, or output) of each such node. As a rule of thumb, nodes are placed on the substrate to reflect the geometry of the task [5, 8, 22]. That way, the connectivity of the substrate is a function of the the task structure.

For example, the sensors of an autonomous robot can be placed from left to right on the substrate in the same order that they exist on the robot (figure 3). Outputs for moving left or right can also be placed in the same order, allowing HyperNEAT to understand from the outset the correlation

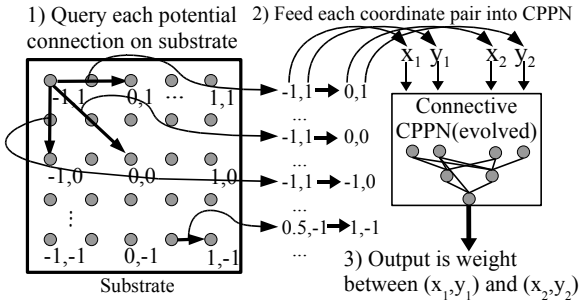


Figure 2: Hypercube-based Geometric Connectivity Pattern Interpretation. A grid of nodes, called the *substrate*, is assigned coordinates such that the center node is at the origin. (1) Every potential connection in the substrate is queried to determine its presence and weight; the dark directed lines in the substrate in the figure represent a sample of connections that are queried. (2) For each query, the CPPN takes as input the positions of the two endpoints and (3) outputs the weight of the connection between them. Thus, *connective CPPNs* can produce regular patterns of connections in space.

of sensors to effectors. In this way, knowledge about the problem can be injected into the search and HyperNEAT can exploit the regularities (e.g. adjacency, or symmetry) of a problem that are invisible to traditional encodings.

In summary, HyperNEAT is a method for evolving ANNs with regular connectivity patterns that uses CPPNs as an indirect encoding. For a full description of HyperNEAT see Stanley et al. [22]. The next section explains how this approach is extended to multiple agents.

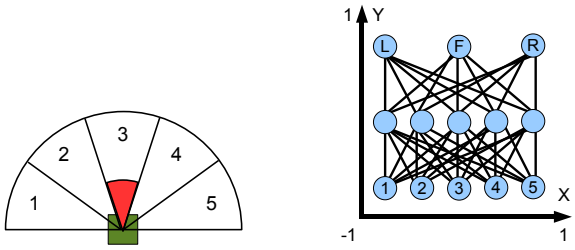
3. APPROACH: Multiagent HyperNEAT

This section begins by exploring how teams of *homogeneous* agents can be evolved with an indirect encoding, and then introduces the Multiagent HyperNEAT approach to evolving a heterogeneous team represented by a single genome.

A homogeneous team only requires a single controller that is copied once for each agent on the team. To generate such a controller, a four-dimensional CPPN with inputs x_1, y_1, x_2, and y_2 (figure 4a) queries the substrate shown in figure 4c, which has five inputs, five hidden nodes, and three output nodes, to determine its connection weights. This substrate is designed to geometrically correlate sensors to corresponding outputs (e.g. seeing something on the left and turning left). Thus the CPPN can exploit the geometry of the agent. However, the agents themselves have exactly the same policy no matter where they are positioned.

Heterogeneous teams are a greater challenge; how can a single CPPN encode a *set* of networks in a pattern, all with related yet varying roles? The main idea is to place the whole set of networks on the substrate and compute their connection weights as *both* a function of their location within each network *and* within the larger pattern of multiple networks. The CPPN then queries all the connection weights in this multiagent substrate (figure 4d), which contains five copies of the homogeneous substrate, one for each agent.

The challenge for such a substrate is to tell the CPPN where one agent stops and another begins; the CPPN could learn this pattern, but it is more effective to tell it where each agent is from the start. For this purpose, two special



(a) Robot (b) Substrate

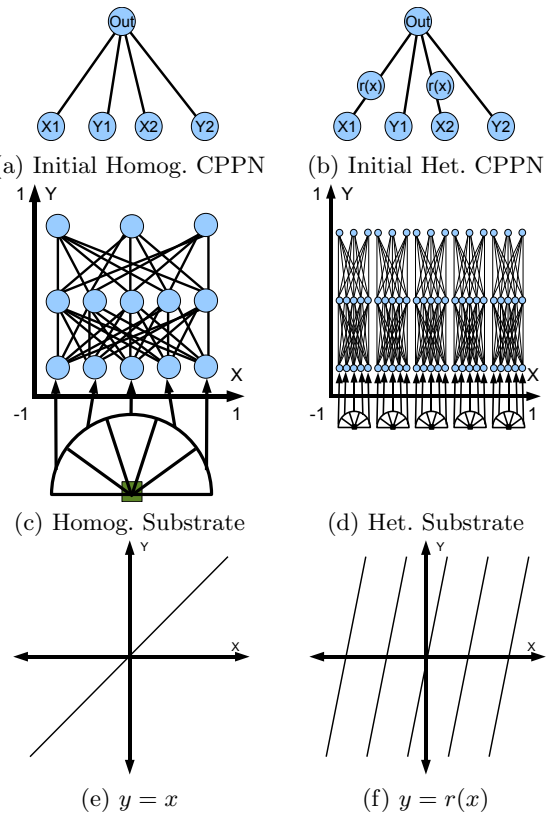
Figure 3: Substrate Configuration. An autonomous robot (a) is equipped with five sensors, spanning a 180° arc in front of it and labeled 1 through 5 from left to right. The substrate that controls the robot (b) is arranged such that the placement of inputs in the neural network corresponds to the physical locations of the sensors on the robot (e.g. the leftmost sensor corresponds to the leftmost output). Similarly, the outputs of the network are related to their effects on the agent and correspond to the sensors (e.g. the left turn output is on the left side of the network and above the leftmost sensor input). Such placement allows the CPPN to easily generate connectivity patterns that respect the geometry of the problem, such as left-right symmetry.

function nodes are added to the initial CPPN so that each receive input from either x_1 or x_2 and output a coordinate frame, $r(x)$, that repeats the same set of coordinates once per agent (figure 4f), thereby telling the CPPN where each node is *within* each individual agent. The CPPN thus sees *both* the coordinate frame $r(x)$ within each agent and the coordinate frame x of the entire team (from inputs x_1 and x_2). In this way, it can simultaneously encode shared patterns within agents (i.e. by basing them on the repeating coordinate frame) and patterns that vary across teams (e.g. agents on the left can mirror the behaviors of agents on the the right through a symmetric function of position). This capability is powerful because generating each agent with the same CPPN means they can share tactics and policies while still exhibiting variation.

In other words, policies are spread across the substrate in a pattern just as role assignment in a human team forms a pattern across a field. However, even as roles vary, many skills are shared, an idea elegantly captured by indirect encodings. This method of encoding a heterogeneous team initially creates a homogeneous team because nothing initially connects x_1 and x_2 to the rest of the CPPN except through $r(x)$. During evolution, however, mutations can connect the absolute x coordinates to the rest of the network, yielding heterogeneous behavior.

Finally, the structure of the CPPN (figure 4b) also allows a team to be *seeded* with the behavior of a single agent, which is a powerful capability if strong single-agent controllers are already available or easily generated. To seed, the x_1 and x_2 inputs in the single agent CPPN (figure 5, left) are diverted to new $r(x)$ nodes, creating a repeating connectivity pattern. All connections originally leaving x_1 and x_2 are then moved so they project from their respective $r(x)$ nodes (figure 5, right). This technique works because the coordinate system defined by $r(x)$ for each agent is *exactly the same* as the a single agent's coordinate system (figure 4e), except that now it repeats several times across x (figure 4f).

The next section describes a multiagent experiment to test this approach.



(a) Initial Homog. CPPN (b) Initial Het. CPPN

(c) Homog. Substrate (d) Het. Substrate

(e) $y = x$ (f) $y = r(x)$

Figure 4: Multiagent HyperNEAT. This figure depicts the CPPNs and substrates that encode multiple agents with HyperNEAT. The CPPN in (a) generates a single controller for a single agent or a homogeneous team of agents. The single controller substrate that is queried by this CPPN is shown in (c). In contrast, the CPPN in (b) encodes heterogeneous teams by sampling the substrate in (d), which is the single substrate (c) copied five times, but compressed horizontally. The $r(x)$ nodes above x_1 and x_2 in the initial heterogeneous CPPN (b) repeat the x coordinate frame (e), duplicating it for each agent while also maintaining a global coordinate system through x_1 and x_2 (f). In this way, the CPPN can create patterns across both the agents' bodies and the team as a whole. If the same point is sampled within any two agents in (d), $r(x)$ will return the same value (though x will not), giving agents on the team their own coordinate frame. Note that CPPNs depicted in (a) and (b) complexify over evolution through the NEAT algorithm.

4. PREDATOR-PREY EXPERIMENT

The aim of the experiment is to demonstrate how GDS can facilitate multiagent learning. Cooperative multiagent predator-prey is a good platform to test this idea because the task is challenging yet easy to understand. While traditional learning techniques offer potential solutions to multiagent problems [16, 28], the authors do not know of any attempt to apply GDS to creating such a team of agents. With GDS, the hope is that tightly coordinated agent policies can be encoded as a pattern with an indirect encoding.

In the version of predator-prey in this paper, agents cannot see their teammates, and because prey run away from nearby predators, it is easy for one predator to undermine another's pursuit by knocking its prey off its path. There-

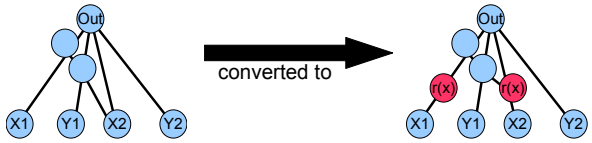


Figure 5: Heterogeneous seeding. Given a CPPN that generates a successful single agent (left), it is possible to create a team of agents based upon it. A heterogeneous team needs to differentiate between team members so the CPPN at left is modified so that the $x1$ and $x2$ inputs feed into two new nodes (shown at right). All connections that originally project from the inputs are changed to project from these new nodes (darkened) instead. This change maintains the coordinate frame of a single agent, while the $x1$ and $x2$ inputs now contain the coordinate frame of the *team*, allowing the CPPN to generate patterns relevant to both.

fore, predators must learn consistent roles that complement those of their allies. At the same time, agents need basic skills in interpreting and reacting to their sensors. Because HyperNEAT creates all the agent ANNs with a GDS approach from the *same* CPPN, it has the potential to balance these delicate ingredients.

4.1 Predators and Prey

Each predator agent on the team is controlled by the ANN in figure 3b. Predators are equipped with five rangefinder sensors spanning a 180° arc that detect prey within 300 units. Their goal is to capture (i.e. intercept) the prey agents by positioning themselves so that a prey is visible to their front sensor and less than 25 units away (this area is shown as a shaded cone in 3a). Predators *cannot* sense each other.

At each discrete moment of time, a predator can turn up to 36° and move up to five units forward. The number of units moved is $5F$, where F is the forward effector output. The predator also turns by $(L - R) * 36°$, where L is the left effector output and R is the right effector output. A negative value is interpreted as a right turn.

Prey agents are programmed to maintain their current location until they are threatened; if there is a predator within 50 units the prey moves in the opposite direction of the closest predator. Prey move at the maximum speed of predator agents. That way, it is impossible for a single predator to catch a prey.

The predator team starts each trial in a line, 100 units apart, facing the prey (figure 6). The environment the agents inhabit is physically unbounded, and each trial lasts $1,000$ time steps. At the end of each trial the team receives a score of $10,000P + (1,000 - t)$, where P is the number of prey captured and t is the time it takes to catch all the prey. If all prey are not captured, t is set to $1,000$. Team fitness is the sum of the scores from two trials on which the team is evaluated. This fitness function encourages the predators to capture all the prey as quickly as possible.

The major challenge for the predators is to coordinate despite their inability to see one another. This restriction encourages establishing *a priori* policies for cooperation because agents thus have little information to infer each others' current states. Such situations are not uncommon. Military units often form plans, split up, and execute complicated maneuvers with little to no contact with each other [6].

4.2 Homogeneous vs. Heterogeneous Policies

To investigate whether role-differentiation helps, teams of cooperative agents can be homogeneous or heterogeneous. Homogeneous teams must rely on their current perceived state to differentiate themselves, which is effective in some tasks [3]. In contrast, heterogeneous teams have more tactical options available because the search does not need to find one global policy that works for all agents in all cases, that is, it can separate the problem among agents. Such separation can be distributed logically across the team (e.g. agents on the left attack prey on the left). Additionally, while the policies may be heterogeneous, they likely should overlap by a significant amount (e.g. all predators know how to turn to face prey).

In three-predator/one-prey predator-prey, Yong and Miikkulainen [28] showed that *coevolved* heterogeneous teams are more effective than homogeneous. This paper takes this result one step further by testing whether larger heterogeneous teams generated by indirect encodings can encode appropriate patterns of behavior. Thus both homogeneous and heterogeneous teams, as described in Section 3, are compared in each experiment.

4.3 Seeding

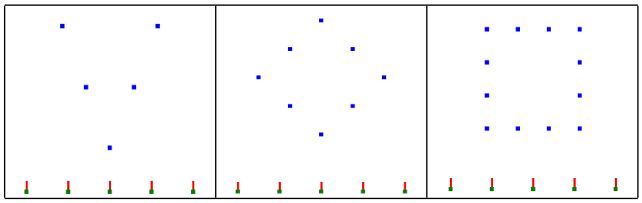
In addition to starting evolution from scratch as normal, this paper also investigates injecting knowledge into the initial search by *seeding* evolution so that the initial population contains variations on a seed genome. While this genome can be from previous evolution or hand-crafted to exploit a particular aspect of the problem, the interesting idea afforded by multiagent HyperNEAT is to seed multiagent learning with the genome of a *single* agent (Section 3). Seeding a team with a single strong agent is effective because a single policy with a set of basic skills is faster and easier to evolve than a whole team.

To verify that seeding in this manner is useful in this domain, both heterogeneous and homogeneous teams are tested with and without evolutionary seeds. The seed is created by evolving a single predator agent that is evaluated only on its ability to chase prey, which is a good starting point for multiagent predators. Strong single agents were typically discovered in less than 50 generations and the best performing agent among them is the seed for the experiments in this paper.

4.4 Prey Formations

Agent teams face two conflicting goals: robust generalization and specialization for efficiency. To balance these goals, while each team is trained on only one of three prey formations (triangle, diamond, and square; figure 6), they are trained on two variations of that formation. Training on only one formation encourages discovering specific tactics to deal with the specific formation, enabling the teams to capture prey more quickly. However, training on multiple variations of the same formation encourages teams to develop robustness to minor changes in that formation. To generate these variations, while the number of prey is constant, their locations are changed by varying the angle or length of the formations.

Each formation creates a significantly different challenge because they not only vary in number of prey, but also in how many prey can be initially seen by each predator. Thus each approach is tested on a variety of multiagent scenarios.



(a) Triangle (b) Diamond (c) Square

Figure 6: Prey Formations. The prey (upper agents) can be arranged in three formations. The predators (lower agents) are always placed in the same evenly-spaced line below the prey. Each formation presents a unique challenge.

4.5 Experimental Parameters

Because HyperNEAT differs from original NEAT only in its set of activation functions, it uses the same parameters [23]. Experiments were run with a modified version of the public domain SharpNEAT package [9]. The size of each population was 150 with 20% elitism. Sexual offspring (50%) did not undergo mutation. Asexual offspring (50%) had 0.96 probability of link weight mutation, 0.03 chance of link addition, and 0.01 chance of node addition. The coefficents for determining species similarity were 1.0 for nodes and connections and 0.1 for weights. The available activation functions were sigmoid, Gaussian, absolute value, and sine, all with equal probability of being added to the CPPN. Parameter settings are based on standard SharpNEAT defaults and prior reported settings for NEAT [21, 23, 25]. They were found to be robust to moderate variation through preliminary experimentation.

5. RESULTS

Performance in this section is measured as the time remaining after capturing all the prey, averaged across each formation variant. Each trial is run for $5,000$ time steps. The maximum (though impossible) score is thus $5,000$ and the minimum score is 0 if all prey were not captured. This measure, which highlights completion of the task, does not necessarily increase with generations like fitness because if a potential solution solves one training example but not the other, it may have to sacrifice performance on the solved example to solve the other. Nevertheless, performance follows a general upward trend over generations and reveals the ultimate quality of solutions.

5.1 Training Performance

Figure 7 shows training performance over generations for teams with homogeneous and heterogeneous policies, with and without seeding, on the formations in figure 6. In all three scenarios, the most successful approach was the seeded heterogeneous team, which outperformed all teams across all configurations. Although the difference between seeded heterogeneous and heterogeneous is only significant in the square formation ($p < 0.001$ after generation 714) and diamond formation ($p < 0.05$ after generation 521), it is significant versus both homogeneous approaches on all formations (eventually at least $p < 0.01$). Also in every formation, unseeded heterogeneous performed second best (eventually at least $p < 0.05$ on all formations), followed by seeded homogeneous ($p < 0.001$ after gen 10 versus homogeneous on triangle, 124 on diamond, and not significant on square) and unseeded homogeneous. Both homogeneous team types could not consistently solve training examples.

A potential question is whether the heterogeneous substrate gains an advantage simply by having more nodes. To check this possibility, all homogeneous experiments were repeated with a substrate with the same number of hidden nodes as the heterogeneous substrate (i.e. a single member of the homogeneous team has 25 nodes, as many as the entire team of heterogeneous agents). The performance of the resulting homogeneous teams was not significantly different, which confirms that the distribution of policies on the substrate is what favors heterogeneous teams.

While the different formations do not change the overall ranking of the policy distributions, the teams did perform differently on them. It turns out that the formations increase in difficulty by number of prey because as the number of prey increases, the viability of picking off one prey at a time decreases; thus the teams require more holistic solutions to capture more prey, widening the gap between heterogeneous and homogeneous teams.

5.2 Generalization

Solutions were tested for their ability to generalize to seven variants of each training formation. Diamond formations vary in length from 100 to 300 units, squares vary in side length from 75 to 225 units, and triangles vary from 0° to 180°. The most general solutions perform well on both training *and* testing, for a total of nine variants of each formation. To make the comparison fair, only the most general solutions produced by each of the twenty runs of evolution are compared. That way reported results indicate the best each approach can do. This method of testing generalization follows Gruau et al. [10] and is designed to compare the best overall individuals.

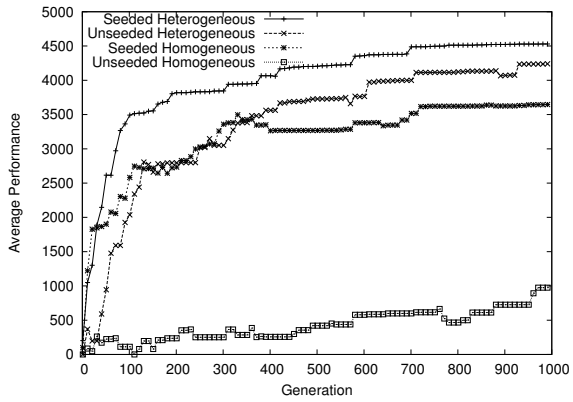
Figure 8 shows that generalization performance is highly correlated with training performance (i.e. the ranking of approaches is the same as in training), which means that heterogeneous roles provide a significant advantage. Only the seeded heterogeneous strategy produces teams that could solve all nine scenarios. The most general teams usually employ the same policy for each variant, although some heterogeneous teams change their policy depending on the specific variant.

5.3 Typical Behaviors

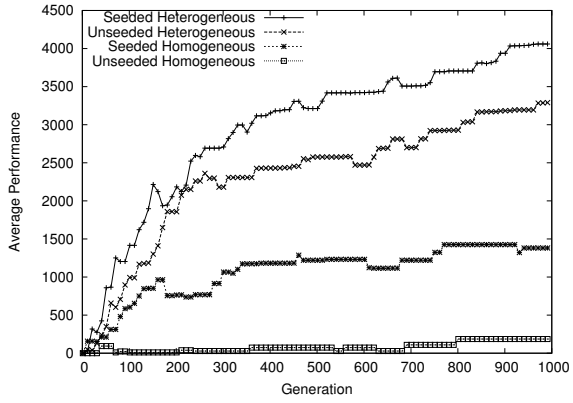
Heterogeneous and homogeneous teams learned significantly different methods of capturing the prey.

The best homogeneous teams rely almost exclusively on every predator finding a prey and chasing it in a circular pattern; if two of these circles overlap one of the predators eventually sees the prey being chased by the other and starts to move toward it. If the predators are well-aligned when one sees the other's prey, they continue to move toward each other and capture both prey; if they are not aligned, one predator abandons its prey to capture the one being chased by the other predator. This policy can be somewhat general, but if there are no other predators nearby, one predator may chase one prey forever. Also, it is inefficient because much time is wasted by running in circles rather than capturing prey.

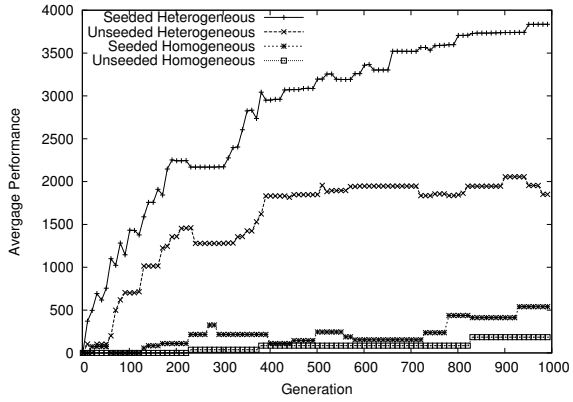
In contrast, heterogeneous agents employ a variety of effective techniques, many exhibiting interesting policy variations based on the team's initial geometric layout, confirming the ability of the multiagent CPPN to encode patterns across the substrate. In one policy, predators hunt for prey



(a) Triangle Formation Performance



(b) Diamond Formation Performance



(c) Square Formation Performance

Figure 7: Comparing Performance of Different Training Methods. The performance of each training method on the three formations is shown, each averaged over twenty runs. In all cases heterogeneous teams significantly outperform homogeneous teams and seeded teams outperform unseeded.

in packs of two or three, commonly teaming with adjacent agents. Upon seeing a prey, they approach from opposite sides and either capture the prey with a pincer attack or trap the prey between them, moving in parallel with the prey until they eventually turn to face and capture it. A second heterogeneous policy is *corralling*, in which predators compress the prey into a tightly-packed cluster and then capture them. Some of the predators move around the perimeter of

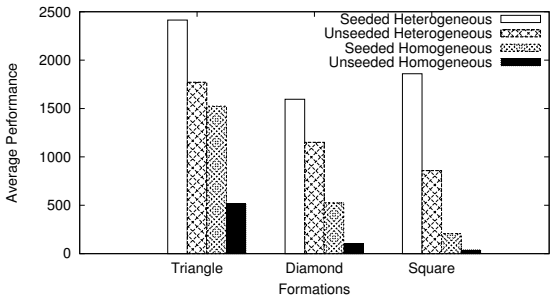


Figure 8: Generalization Performance. Average performance is shown for each approach on the nine variants of each prey formation, averaged over twenty runs. Heterogeneous methods generalize significantly better than homogeneous (at least $p < 0.01$ in all cases except unseeded heterogeneous versus seeded homogeneous on triangle) and seeding produces significantly better performance than starting from scratch (at least $p < 0.05$ in all cases). The main conclusions is that both heterogeneity and seeding afford significant advantage in generalizing in this domain.

the prey, forcing them to run toward the center of the formation, while others form a *fence* and slowly advance toward the cluster. Corralling is usually symmetric and a result of policy mirroring, which means that predators from the east and west starting positions rotate around the prey in opposite directions to maximize efficiency. A third policy deploys *posts*, which are usually the predators on the left and right starting positions, who serve as stationary traps. Mobile predators then chase prey into the posts. Almost all heterogeneous solutions rely on one or a combination of these policies, although a few exhibit behaviors similar to homogeneous solutions. Videos of agent behaviors are available at `http://eplex.cs.ucf.edu/multiagentHyperNEAT/`.

While seeded and unseeded solutions employ generally the same tactics, the major difference is that unseeded teams often include agents that serve no useful purpose. Such agents spin in circles, move away from the prey, or just sit still. This inefficiency explains the overall lower performance of unseeded strategies, confirming that starting with basic skills provides a solid foundation for eventual differentiation.

6. DISCUSSION AND FUTURE WORK

The result that heterogeneous teams significantly outperform homogeneous in training and generalization demonstrates that the HyperNEAT GDS approach successfully encodes heterogeneous roles that contribute to superior performance. The ability to encode patterns of behavior across a team is critical to success in multiagent learning and thereby addresses a major challenge in the field. The HyperNEAT method allows team behavior to be represented as variation on a theme encoded in a *single genome*, meaning that key skills need not be rediscovered for separate agents. Furthermore, because multiagent policies are represented by a CPPN, they are assigned to separate agents as a function of their relative geometry, while simultaneously exploiting the agents' internal geometries.

Seeding evolution was also beneficial. This capability captures the idea that real-life teams (e.g. in soccer) often share a critical basic skill set that can be learned faster by an individual agent than an entire team. While HyperNEAT natu-

rally encodes variations on a theme, finding the right underlying theme can initially be challenging. Seeding bootstraps the process, providing a mechanism to inject domain knowledge. In the future, the sophistication of team behavior can be increased by evolving seeds on many subgoals, such as running, passing, shooting, and defending in soccer, which can be duplicated across the entire team and then allowed to vary by role.

A desirable property of multiagent systems is scalability [16], and an exciting future extension to Multiagent HyperNEAT is the ability to change the team size *without* further evolution. Because HyperNEAT CPPNs encode policies as a function of their positions, individuals added to the substrate at new positions (e.g. by compressing each ANN to fit in more ANNs) should naturally receive gracefully interpolated policies. Also, in the same way, the team can dynamically reassign policies when an agent is lost or damaged simply by shifting their positions on the substrate.

The main result is that combining seeding and heterogeneity produces the most effective teams, confirming the power of GDS to encode multiagent behavior through information reuse and pattern generation. In the future it will be important to compare the relative strengths and weaknesses of such an approach to other multiagent learning approaches. The contribution of this paper is to establish GDS as a credible new option in the field of multiagent learning.

7. CONCLUSIONS

This paper introduced a new approach to encoding teams of agents by using the HyperNEAT method. Results in a predator-prey task show that a team of heterogeneous agents that has been seeded with a strong single agent can learn genuinely cooperative behavior with differentiated roles. By applying GDS to multiagent learning, teams are generated that quickly and easily learn a set of common skills as well as a pattern of cooperative behavior relative to the geometry of the team. Such teams can also generalize to unseen formations. Thus the results in this paper establish that multiagent learning benefits from the indirect encoding of GDS. With further research, it is possible that multiagent learning is a "killer app" for which GDS has been searching.

Acknowledgments

This research was supported in part by DARPA under grant HR0011-08-1-0020 (Computer Science Study Group).

8. REFERENCES

[1] P. J. Bentley and S. Kumar. The 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.

[2] J. Bongard. Reducing Collective Behavioural Complexity through Heterogeneity. *Artificial Life VII: Proceedings of the Seventh International Conference on Artificial Life*, 2000.

[3] B. D. Bryant and R. Miikkulainen. Neuroevolution for adaptive teams. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003)*, volume 3, pages 2194–2201, 2003.

[4] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

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

[6] T. N. Dupuy. *The Evolution of Weapons and Warfare*. Da Capo, New York, NY, USA, 1990.

[7] P. Eggenberger. Evolving Morphologies of Simulated 3d Organisms Based on Differential Gene Expression. *Fourth European Conference on Artificial Life*, 1997.

[8] J. Gauci and K. 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.

[9] C. Green. SharpNEAT homepage. `http://sharpneat.sourceforge.net/`, 2003–2008.

[10] F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 81–89. MIT Press, 1996.

[11] J. Hu and M. P. Wellman. Multiagent reinforcement learning: theoretical framework and an algorithm. In *Proc. 15th International Conf. on Machine Learning*, pages 242–250. Morgan Kaufmann, San Francisco, CA, 1998.

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

[13] S. Luke and L. Spector. Evolving graphs and networks with edge encoding: Preliminary report. In J. R. Koza, editor, *Late-Breaking Papers of Genetic Programming 1996*. Stanford Bookstore, 1996.

[14] A. P. Martin. Increasing genomic complexity by gene duplication and the origin of vertebrates. *The American Naturalist*, 154(2):111–128, 1999.

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

[16] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 3(11):383–434, November 2005.

[17] L. Panait, R. Wiegand, and S. Luke. Improving coevolutionary search for optimal multiagent behaviors. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 653–658, 2003.

[18] M. A. Potter, K. A. De Jong, and J. J. Grefenstette. A coevolutionary approach to learning sequential decision rules. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, 1995.

[19] K. Sims. Evolving 3D morphology and behavior by competition. pages 28–39. MIT Press, Cambridge, MA, 1994.

[20] K. 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] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation Special Issue on Evolutionary Computation and Games*, 9(6):653–668, 2005.

[22] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based indirect encoding for evolving large-scale neural networks. *Artificial Life*, 2008. To appear.

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

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

[25] K. O. Stanley and R. Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artifical Inetlligence Research*, 21:63–100, 2004.

[26] P. Stone and M. Veloso. Multiagent Systems: A Survey from a Machine Learning Perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[27] R. P. Wiegand. *An analysis of cooperative coevolutionary algorithms*. PhD thesis, George Mason University, Fairfax, VA, USA, 2004. Director-Kenneth A. Jong.

[28] C. Yong and R. Miikkulainen. Coevolution of role-based cooperation in multi-agent systems. Technical Report AI07-338, The University of Texas at Austin Department of Computer Sciences, 2007.