Evol. Intel. (2013) 6:1–26
DOI 10.1007/s12065-012-0086-3

RESEARCH PAPER

Scalable multiagent learning through indirect encoding of policy geometry

David B. D'Ambrosio · Kenneth O. Stanley

Received: 8 March 2012 / Revised: 7 November 2012 / Accepted: 28 December 2012 / Published online: 18 January 2013
© Springer-Verlag Berlin Heidelberg 2013

Abstract Multiagent systems present many challenging, real-world problems to artificial intelligence. Because it is difficult to engineer the behaviors of multiple cooperating agents by hand, *multiagent learning* has become a popular approach to their design. While there are a variety of traditional approaches to multiagent learning, many suffer from increased computational costs for large teams and the *problem of reinvention* (that is, the inability to recognize that certain skills are shared by some or all team member). This paper presents an alternative approach to multiagent learning called *multiagent HyperNEAT* that represents the team as a *pattern* of policies rather than as a set of individual agents. The main idea is that an agent's location within a canonical team layout (which can be physical, such as positions on a sports team, or conceptual, such as an agent's relative speed) tends to dictate its role within that team. This paper introduces the term *policy geometry* to describe this relationship between role and position on the team. Interestingly, such patterns effectively represent up to an infinite number of multiagent policies that can be sampled from the policy geometry as needed to allow training very large teams or, in some cases, scaling up the size of a team without additional learning. In this paper, multiagent HyperNEAT is compared to a traditional learning method, multiagent Sarsa(λ), in a predator–prey domain, where it demonstrates its ability to train large teams.

D. B. D'Ambrosio (✉) · K. O. Stanley
Department of Electrical Engineering and Computer Science, University of Central Florida, 4000 Central Florida Blvd., Orlando, FL 32816-2362, USA
e-mail: ddambro@eecs.ucf.edu

K. O. Stanley
e-mail: kstnaley@eecs.ucf.edu

Keywords Multiagent learning · Indirect encoding · HyperNEAT · Neural networks

1 Introduction

Cooperative multiagent learning focuses on training groups of autonomous agents to work together to solve problems. By intelligently dividing responsibility among the agents, such teams can solve difficult problems that may be prohibitive for a single agent [78, 100, 107]. Researchers across artificial intelligence are interested in multiagent systems because they apply to real world problems and the challenge of controlling and coordinating multiple interacting agents provides a compelling opportunity to demonstrate the promise of sophisticated learning techniques [8, 66, 100].

At present, two leading approaches to multiagent learning are cooperative coevolutionary algorithms (CCEAs; [31, 67, 68, 73, 74])and multiagent reinforcement learning (MARL; [10, 17, 47, 102]). At heart, these approaches pose multiagent learning as an extension of single-agent learning in the sense that teams are trained as multiple instances of the single-agent learning problem that are also rewarded for effectively cooperating. In contrast, this paper provides a novel perspective on how to represent and train a team of agents based on discovering how their roles relate to each other.

An interesting property of real life teams is that the behaviors and policies of team members tend to be dictated by their canonical position within the team. The result is that the policies (what an agent should do given a certain situation) of agents on a team are often distributed in a pattern according to their positions. In other words, the

team has a *policy geometry* that dictates how each member should behave based on their location within the team. For example, in a soccer (that is, football) team (Fig. 1), 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, agents tend to *share* common skills; in the soccer example, all players know how to pass and kick.

Note that these positions need not be related to the physical location of the team members. For example, in a team of collaborating robots that move at different speeds, their "position" along the dimension of speed defines a *conceptual* policy geometry that similarly can dictate the roles of the robots (that is, slower robots should perform different tasks than faster ones). An intriguing implication of the idea of policy geometry for multiagent learning is that, rather than learning the policies of individual agents, the training method can learn how the pattern of policies relates to the geometry of the team and *derive* the policies of individual agents based on that relationship.

To see the benefits of this perspective, consider that in traditional approaches, the complexity of the multiagent learning problem increases with the size of the team because the number of possible team states grows with the number of agents, as does the uncertainty of individual agents, which limits the size of teams that can be trained [14, 67]. However, representing multiagent teams as a *pattern of policies*, that is, as a set of related policies, rather than as individual agents, eliminates this problem because all that needs to be learned and represented is the pattern. Even more interesting is that such a pattern effectively represents up to an infinite number of multiagent policies that can be sampled from the policy geometry as needed; thus multiagent teams represented in this way have no *a priori* bound on their team size. This capability means that large teams can be trained. It also provides a heuristic for scaling the size of a team that was trained to complete a task with a specific number agents up and down dynamically, depending on the number of agents available when the task is attempted in the real world.

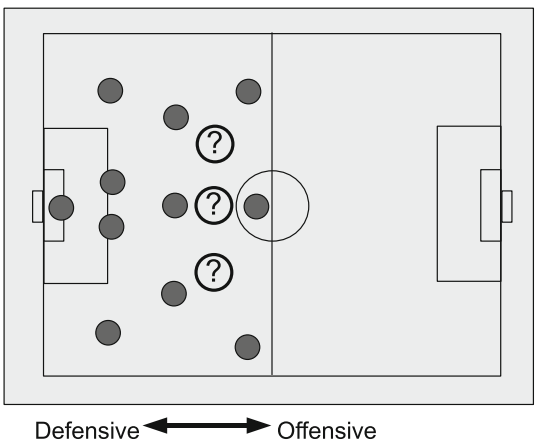


Fig. 1 Role as a function of geometry. In the soccer team in the picture, the defensive to offensive dimension of variation spans from *left* to *right*. The *question marks* represent a hypothetical new line of players whose roles can be inferred from their positions. An important observation is that the policies are distributed in a *pattern* in space

While homogeneous and heterogeneous learning are usually separated conceptually in discussions of multiagent learning [67, 99, 114], another important insight is that heterogeneity can alternatively be viewed as a *continuum* that begins at pure homogeneity and ends at radical differentiation. For example, in this view, it is possible for a team to be nearly homogeneous but not completely. In such a team, to a large extent, many skills and behaviors are shared among agents on the team even though individuals possess their own unique tendencies, suggesting the possibility to avoid redundantly discovering and representing common skills. Such intermediate configurations between pure homogeneity and extreme heterogeneity are common in human teams. For example, in soccer, players *share* many fundamental abilities, from low-level universal human faculties such as visual recognition to high-level skills such as passing and dribbling the ball. In fact, it is difficult to think of a kind of team that is purely heterogeneous (that is, no team members share any traits or skills whatsoever).

This view of heterogeneity as extending from pure homogeneity exposes a potential problem with approaches that treat heterogeneous agents as separately learned entities. This *problem of reinvention* is that many of the shared skills that often should dominate the policies of every agent in fact must be reinvented separately for each agent. Thus a more principled approach should be able to represent the commonalities as a regularity that only needs to be discovered and encoded once. The novel approach in this paper shows how such team encoding is possible through learning policy geometry.

In effect, this paper introduces a method that both addresses the problem of reinvention and provides a way to overcome the computational obstacles to scaling the size of multiagent teams by representing them as patterns of policies rather than as individual agents. To implement this idea, *hypercube-based neuroevolution of augmenting topologies* (HyperNEAT), an approach to evolving artificial neural networks (ANNs) that has demonstrated success in single agent control and in encoding large, scalable neural networks [18, 35, 96, 112], is extended to encode *patterns of ANNs distributed across space* with a single genome. The spatial distribution of ANNs matches with the locations (physical or conceptual) of agents on the team, thereby allowing HyperNEAT to learn a *pattern of policies* (that is, the policy geometry), all generated from the same genome. In this way, HyperNEAT can reuse critical

information by learning the ways in which roles relate to one another to conquer the problem of reinvention and scale teams to new sizes by sampling from the policy geometry. The idea of applying HyperNEAT to multiagent learning in this way was first raised in conferences papers by D'Ambrosio and Stanley [26], D'Ambrosio et al. [25], and later Knoester et al. [52], but is significantly expanded and comprehensively investigated in this paper for the first time.

To demonstrate its promise, the multiagent HyperNEAT method is tested in a multiagent predator–prey domain in which a team of multiple predators that cannot see each other is trained 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 the more traditional multiagent Sarsa(λ) reinforcement learning technique, focusing on how both methods fare with very large team sizes. Furthermore, smaller HyperNEAT-trained teams are scaled *after training* to much larger sizes, up to 1,024 agents, that still coordinate seamlessly, thereby establishing the ability to scale afforded by learning the policy geometry.

Thus the main conclusion is that by representing teams as a pattern of policies rather than as individual agents, HyperNEAT can train much larger effective multiagent teams than have heretofore been possible and provides a heuristic to scale already-trained teams to larger sizes. Additionally, by viewing the team as a point on the continuum of heterogeneity, rather than as purely heterogeneous or homogeneous, HyperNEAT is able to vary existing singleton policies into effective team-wide strategies. In contrast, multiagent Sarsa(λ) does not have access to such tools and is only able to find piece-wise solutions for smaller teams. Thus, the new ideas in multiagent HyperNEAT contribute a novel practical advantage. In the future, by exploiting policy geometry, this advantage may also be possible to extend to more traditional approaches such as multiagent Sarsa(λ) as well.

The paper begins with a review of cooperative multiagent learning, NEAT, and HyperNEAT in the next section. The multiagent HyperNEAT approach is then detailed in Sect. 3. Section 4 describes and presents results in a predator–prey domain. Section 6 then discusses the implications of these results and outlines future work, followed by conclusions in Sect. 7.

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, creating the opportunity for real-world applications such as room clearing, pursuit [27], and synchronized motion [77]. In cooperative multiagent learning, which is reviewed in this section, agents are trained to work together to accomplish a task, usually by one of several alternative methods. Teams can sometimes share a homogeneous control scheme, which means that all agents have the same control policy and thus only one policy is learned. However, it has been shown [121] that teams of agents with heterogeneous behaviors can solve tasks that homogeneous teams cannot; thus this paper will focus mainly on teams with heterogeneous policies but homogeneous physical capabilities.

2.1.1 Traditional approaches

There are two major classes of approaches to multiagent learning: multiagent reinforcement learning (MARL; [23, 58, 87, 97]) and cooperative coevolutionary algorithms (CCEAs; [31, 67, 72]). While these approaches are mainly the focus of separate communities, Panait et al. [71] noted recently that they share a significant common theoretical foundation. One key commonality is that they break the learning problem into separate roles that are semi-independent and thereby learned separately through interaction with each other. Although this idea of separating multiagent problems into parts is popular, it does create challenges for certain desirable objectives such as scaling to larger team sizes, which is a focus in this paper. The problem is that when individual roles are learned separately, there is no representation of how roles relate to the team structure and thus it is difficult to assign new roles automatically to new individuals; they must somehow be derived through interactions or inferences [9, 15]. Nevertheless, these approaches have produced significant insight into multiagent learning, and are reviewed in this section.

Multiagent reinforcement learning encompasses several specific techniques based on off-policy and on-policy temporal difference learning [10, 17, 47, 102]. The basic principle that unifies MARL techniques is to identify and reward cooperative or beneficial states and actions among a team of agents to encourage their repetition, eventually resulting in an effective team after a number of repetitions [13, 67]. For example, if a group of predator agents is tasked with capturing a prey, agents are *all* rewarded at least in part when the prey is captured. The reward is a signal that their recent actions were useful and should be repeated with greater probability. Conversely, actions that are not rewarded or that are punished are less likely to be repeated. Rewards can be given globally (to the entire

team) or locally (to single agents or groups of agents) and agents may update their own policies or maintain a single group policy depending on the specific method chosen.

One reason that MARL is attractive is because it is possible to prove convergence to Nash equlibria in some situations [46, 87]. Yet such guarantees rely on having complete or at least significantly large amounts of information about the world state, and lacking such information is often what makes multiagent domains challenging [85]. Even in scenarios with sufficient state information, the number of possible states the team can grow with the number of agents, making the full state space difficult to approximate and increasingly uncertain. Additionally, there is also the credit assignment problem [61], wherein it is not always possible for the individual agents to associate the rewards with the correct actions. Thus experimenters must carefully identify the correct reward states manually to minimize this uncertainty. Nevertheless, MARL has provided several successes including intrusion detection [84] and trading strategies [56].

The other major approach, CCEAs, is an established evolutionary method for training teams of agents that must work together [31, 48, 67, 68, 73]. The main idea is to maintain one or more populations of candidate agents, evaluate them in groups, and guide the creation of new candidate solutions based on their joint performance. An important distinction among coevolutionary methods is the population model chosen, which can range from a single-population from which all team members are drawn [12, 76] to separate populations for each agent on the team [73]. Recent work by Panait et al. [69] CCEA performance is enhanced by keeping an archive of informative collaborators, an idea that is also relevant to MARL.

In cooperative coevolution, agents are explicitly rewarded for their cooperative abilities and multiple combinations may be evaluated in the same population, potentially leading to more robust solutions. However, depending on the population model employed, trade-offs between specialization and skill sharing must be made. In the single population model, genetic information is easily spread among the entire population; thus if an agent finds a good general strategy other agents can potentially adopt it. However, with only one population it is difficult for agents to specialize to specific roles [118]. In contrast, multiple populations encourage specialization, yet the separate populations must reinvent basic, useful policies. Additionally, by evaluating agents with many different teams, depending on the evaluation scheme there is a risk of encouraging too much generalization [70], which means the resulting team may not be the best possible. Another significant problem is that adding more agents to the team results in a significant increase in computational complexity; the need for more sampling and potentially more populations and can significantly impact the ability to optimize the performance of the final team [62]. The approach in this paper addresses this problem with training large teams.

Both CCEAs and MARL face the *problem of reinvention*. That is, because agents are treated as separate subproblems they must usually separately discover and represent all aspects of the solution, even though there may be a high degree of overlapping information among the policies of each agent. CCEAs commonly separate agents into different populations, creating strict divisions among agents, and in MARL methods, each agent may learn a separate value function based upon individual experiences. There have been attempts to address the problem of reinvention such as introducing existing agents that "train" new agents [75, 107] or implementing specially designed genetic operators [42]. However, an intriguing alternative is to exploit the *continuum of heterogeneity*, which means distributing shared skills optimally among the agents and only representing such skills once. At the same time, unique abilities could be isolated and assigned appropriately. The method in this paper addresses the problem of reinvention by finding the right point on the continuum of heterogeneity.

There are several extensions to these basic approaches that attempt to make the problem of multiagent learning more tractable. Layered learning [45, 98] takes inspiration from multiagent learning in the sense that a complex problem is broken into smaller sub-problems that should be easier to learn. In layered learning, the policy of each agent is first broken up by the experimenter into subtasks through a hierarchical decomposition. These subtasks are then arranged into layers based on their interdependencies. Subtasks are learned independently, in layer order. That is, subtasks that do not depend on other subtasks are learned first, followed by subtasks that depend on those subtasks and so on. When all subtasks are learned they are combined into a single policy for the agent. A possible downside to this approach is that when the subtasks are trained independently there may be inconsistencies between the subdomain in which they were trained and the actual domain to which they will be applied. Concurrent layered learning [117] addresses this issue by allowing lower layers to continue to learn when higher layers are being trained. Both versions of layered learning have proven successful in many domains, particularly Robocup soccer. However, both methods also require significant effort by the researcher to properly decompose the agent policies and to design sub-domains that effectively train the subtasks. Additionally, both methods restrict the search space of the algorithm, which can speed up search, but also restricts the types of solutions and strategies employed. The approach in this paper avoids the up-front effort of dividing tasks by hand while also bypassing the need for such restrictions.

Another technique that can benefit multiagent learning is transfer learning [106, 108], wherein agents trained in one domain (source task) can be transferred to another domain (target task) while maintaining and exploiting knowledge gained in the original domain. If the target task is similar enough to the source task (that is, the state-action space is the same) teams may immediately be applicable and thereby simply continue learning in the new domain. However if the domains are very different, a mapping must be constructed between the two tasks [109]. In this paper, teams of agents are trained at one size and then transferred to another size, which has been explored previously [109]. However, unlike such previous work, the approach in this paper often does not require retraining to perform tasks at the new team sizes.

In this paper, multiagent Sarsa(λ), an on-policy MARL approach [101, 104] is compared to multiagent HyperNEAT. Sarsa(λ) is ideal for this comparison for several reasons. In terms of computational complexity, training large teams with CCEAs would be too expensive for large teams due to the combinatorial properties of team evaluations; Sarsa(λ) does not have this combinatorial problem. Additionally, it has preformed well in high-profile multiagent tasks [101]. Finally, as mentioned earlier, research indicates theoretical similarities between MARL and CCEAs, thus a comparison against one serves as a comparison against both.

2.1.2 Alternative techniques

Breaking the team into separate parts is not the only way to distribute policies. This section reviews several alternative approaches.

One such approach is to optimize a single, monolithic controller that takes inputs from all the agents and outputs the actions of all the agents. Such consolidation allows information sharing but generally increases the dimensionality of the search significantly, while ignoring separability [121]. Such a model also assumes the existence of global, instantaneous communication, which is infeasible in most real world scenarios. A related approach is to directly encode several disconnected policies in a single monolithic description [6, 59, 62], which gains separability at the expense of information sharing. In both cases, adding more agents to the teams causes large increases in the search space, especially in the single controller case.

One solution to reduce dimensionality in either case without combining multiple individuals is to assign the same homogeneous control system to each agent [4, 11]. 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. This approach has produced significant results in swarm robotics, such as a team of four connected robots that exhibit coordinated trajectories [4]. However, Yong and Miikkulainen [121] show that in predator–prey tasks with three predators and one prey, heterogeneous teams learn more effective strategies than homogeneous ones. Thus, while a homogeneous team may be easier to train and scale, there are limits to what it can do [114]: Pure homogeneity is only a single point on the continuum, while heterogeneity is the continuum.

In summary, there are many challenges faced by multiagent learning, and choosing one method over another generally leads to trade-offs among several competing factors. However, in almost all cases, adding more agents to the team greatly impacts the efficiency of search and, as a result, the overall quality of the resulting multiagent team.

2.2 Evolution and indirect encodings

In the context of reinforcement learning problems (such as in multiagent learning), an interesting property of evolutionary computation (EC) is that it is guided by a fitness function rather than from an error computation derived from a reward prediction. The independence of the fitness function from direct error computation has encouraged much experimentation with alternative representations because representations in EC do not need to support an algorithm for optimizing error. Such freedom has led to the advent of innovative representations for neural networks and also to novel methods for encoding complex structures, as described in this section.

The specific subfield of EC that is implemented in this paper is called neuroevolution (NE), which employs EC to create artificial neural networks (ANNs) [32, 120]. In this approach, the phenotype is an ANN and the genotype is an implementation-dependent representation of the ANN. Assuming that the representation is sufficiently robust, NE can evolve any type of ANN, including recurrent and adaptive networks [79, 88]. Early attempts at NE used fixed-topology models that were designed by the experimenter [64]. In the fixed-topology approach, the genotype is simply an array of numbers that represented the weights of each connection in the network. However, this approach is also restrictive because the solution may be difficult to discover or may not exist at all in the chosen topology. Thus new techniques that allowed evolving both connection weights and network topology were developed [44, 55, 90]. One such method, neuroevolution of augmenting topologies or NEAT, which is described next, has proven successful and serves as the foundation for the multiagent learning approach introduced in this paper.

2.2.1 Neuroevolution of augmenting topologies (NEAT)

The NEAT method was originally developed to evolve ANNs to solve difficult control and sequential decision tasks [90, 92, 94]. In this paper, it is significantly extended to evolve the representation of teams of agents. Nevertheless, the basic principles of NEAT, reviewed in this section, still supply the foundation of the approach.

Traditionally, ANNs evolved by NEAT control agents that select actions based on their sensory inputs. NEAT is unlike many previous methods that evolved neural networks, that is, *neuroevolution* methods, which historically evolved either fixed-topology networks [37, 80], or arbitrary random-topology networks [3, 39, 120]. Instead, NEAT begins evolution with a population of small, simple networks and increases the complexity of the network topology into diverse species over generations, leading to increasingly sophisticated behavior. A similar process of gradually adding new genes has been confirmed in natural evolution [60, 116] and shown to improve adaptation in a few prior evolutionary [2] and neuroevolutionary [41] approaches. However, a key feature that distinguishes NEAT from prior work in evolving increasingly complex structures is its unique approach to maintaining a healthy diversity of structures of different complexity simultaneously, as this section reviews. This approach has proven effective in a wide variety of domains [1, 93, 95, 111]. Complete descriptions of the NEAT method, including experiments confirming the contributions of its components, are available in Stanley and Miikkulainen [90, 92] and Stanley et al. [94].

The NEAT method is based on three key ideas. First, to allow network structures to increase in complexity over generations, a method is needed to keep track of which gene is which. Otherwise, it is not clear in later generations which individual is compatible with which in a population of diverse structures, or how their genes should be combined to produce offspring. NEAT solves this problem by assigning a unique *historical marking* to every new piece of network structure that appears through a structural mutation. The historical marking is a number assigned to each gene corresponding to its order of appearance over the course of evolution. The numbers are inherited during crossover unchanged, and allow NEAT to perform crossover among diverse topologies without the need for expensive topological analysis.

Second, NEAT speciates the population so that individuals compete primarily within their own niches instead of with the population at large. Because adding new structure is often initially disadvantageous, this separation means that unique topological innovations are protected and therefore have the opportunity to optimize their structure without direct competition from other niches in the population. NEAT uses the historical markings on genes to determine to which species different individuals belong.

Third, many approaches that evolve network topologies and weights begin evolution with a population of random topologies [39, 120]. In contrast, NEAT begins with a uniform population of simple networks with no hidden nodes, differing only in their initial random weights. Because of speciation, novel topologies gradually accumulate over evolution, thereby allowing diverse and complex phenotype topologies to be represented. No limit is placed on the size to which topologies can grow. New nodes and connections are introduced incrementally as structural mutations occur, and only those structures survive that are found to be useful through fitness evaluations. In effect, then, NEAT searches for a compact, appropriate topology by incrementally adding complexity to existing structure.

The next section reviews the HyperNEAT extension to NEAT that is itself extended in this paper to generate multiagent teams.

2.2.2 CPPNs and HyperNEAT

A key similarity among many neuroevolution methods, including NEAT, is that they employ a *direct* encoding, that is, each part of the solution's representation maps to a single piece of structure in the final solution. For example, in NEAT, the genome is a list of connections and nodes in the neural network in which each item corresponds to exactly one component in the phenotype. Yet direct encodings impose the significant disadvantage that even when different parts of the solution are similar, they must be encoded and therefore discovered separately. This challenge is related to the problem rediscovery in multiagent systems: After all, if individual team members are encoded by separate genes, even if a component of their capabilities is shared, the search algorithm has no way to exploit such a regularity. Thus this paper leverages the power of *indirect* encoding instead, which means that the description of the solution is compressed such that information can be reused, allowing the final solution to contain more components than the description itself.

For example, if a hypothetical solution ANN required all weights to be set to 1.0, NEAT would separately have to discover that each such weight must be 1.0 whereas an indirect encoding could instead discover that all weights should be the same value. Indirect encodings are often motivated by *development* in biology, in which the genotype (DNA) maps to the phenotype (the living organism) indirectly through a process of growth [5, 57, 91]. Indirect encodings are powerful because they allow solutions to be represented as a pattern of policy parameters, rather than

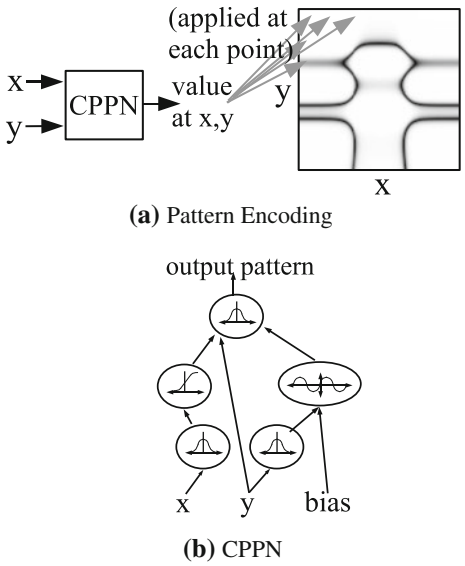
requiring each parameter to be represented individually. This capability is the focus of the field called *generative and developmental systems* [5, 7, 30, 43, 57, 63, 86, 89, 91].

HyperNEAT, reviewed in this section, is an extension of NEAT that allows it to benefit from indirect encoding. HyperNEAT has become a popular neuroevolution method in recent years and is proven in a wide range of domains such as board games [33–36], adaptive maze navigation [79], quadruped locomotion [21], keepaway soccer [112, 113] and a variety of others [18–20, 22, 28, 40, 96, 119]. For a full description of HyperNEAT see Stanley et al. [96] and Gauci and Stanley [35].

In HyperNEAT, NEAT is altered to evolve an indirect encoding called compositional pattern producing networks (CPPNs [89]) *instead* of ANNs. CPPNs are a high-level abstraction of the development process in nature, intended to approximate its representational power without the computational cost. The idea is that regular patterns such as those seen in nature can be approximated at a high level by *compositions of functions*, wherein each function in the composition loosely corresponds to a canonical event in development. For example, a Gaussian function is analogous to a symmetric chemical gradient. Each such component function also creates a novel geometric *coordinate frame* within which other functions can reside. For example, any function *of* the output of a Gaussian will output a symmetric pattern because the Gaussian is symmetric. In this way, the Gaussian is a coordinate frame like a chemical gradient in natural development that provides a context for growing symmetric structures.

The appeal of this encoding is that it allows a representation akin to developmental processes to be encoded as networks of simple functions (that is, CPPNs), 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 development rather than of brains. Also, unlike other artificial developmental encodings, CPPNs do not require an explicit simulation of growth or local interaction, yet still exhibit their essential representational capabilities [89].

Specifically, CPPNs produce a phenotype that is a function of n dimensions, where n is the number of dimensions of the desired solution, for example, $n = 2$ for a two-dimensional image. For each coordinate in that space, its level of expression is output by the CPPN, which encodes the phenotype. Figure 2 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 [24], CPPNs are also universal function approximators. Thus a CPPN can encode any pattern within its n-dimensional space.



(a) Pattern Encoding

(b) CPPN

Fig. 2 CPPN encoding. **a** The CPPN 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 the CPPN, the result is a spatial pattern, which can be viewed as a phenotype whose genotype is the CPPN. **b** Internally, the CPPN is a graph that determines which functions are connected. As in an ANN, the connections are weighted such that the output of a function is multiplied by the weight of its outgoing connection. The CPPN in **b** actually produces the pattern in **a**

The appeal of the CPPN as an indirect encoding is that it can compactly encode patterns with regularities such as symmetry, repetition, and repetition with variation [82, 83, 89]. For example, simply by including a Gaussian function, which is symmetric, the output pattern can become symmetric. A periodic function such as sine creates segmentation through repetition. Most importantly, *repetition with variation* (for example, the fingers of the human hand) is easily discovered by combining regular coordinate frames (for example, sine and Gaussian) with irregular ones (for example, the asymmetric x-axis). For example, a function that takes as input the sum of a symmetric function and an asymmetric function outputs a pattern with imperfect symmetry. In this way, CPPNs produce regular patterns with subtle variations reminiscent of many seen in nature. The potential for CPPNs to represent patterns with natural motifs has been demonstrated in several studies [89] including an online service on which users collaboratively breed patterns represented by CPPNs [82, 83].

The main idea in HyperNEAT is that CPPNs can also naturally encode *connectivity patterns* [33–35, 96, 112]. That way, NEAT can evolve CPPNs that represent large-scale ANNs with their own symmetries and regularities. This capability will prove essential to encoding multiagent policy geometries in this paper because it will ultimately allow connectivity patterns to be expressed as a function of team geometry, which means that a smooth gradient of

policies can be produced across possible agent locations. The key insight in HyperNEAT is that 2*n*-dimensional spatial patterns are *isomorphic* to connectivity patterns in *n* dimensions, that is, in which the coordinate of each endpoint is specified by *n* parameters, which means that CPPNs can express both spatial *and* connectivity patterns with the same kinds of regularities.

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 (Fig. 3). By querying every possible connection among a set of points in this manner, a CPPN can produce an ANN, wherein each queried point is a neuron position. Because the connections are produced by a function of their endpoints, the final structure is a product of the geometry of these points and the CPPN can thus exploit the relationships between them in the network it encodes. In effect, the CPPN is painting a pattern on the inside of a four-dimensional hypercube that is interpreted as the 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 (that is, 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 [20, 21, 34, 35, 96, 112]. That way, the connectivity of the substrate is a function of 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

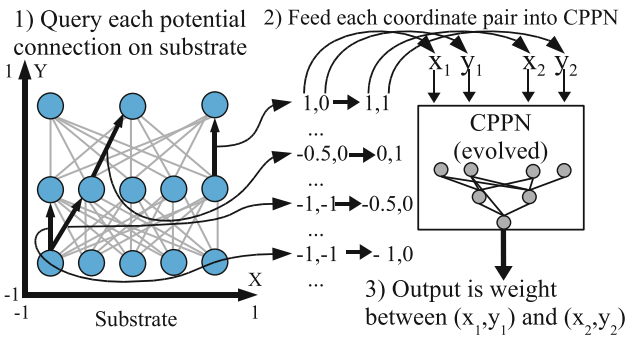


Fig. 3 Hypercube-based geometric connectivity pattern interpretation. A collection of nodes, called the *substrate*, is assigned coordinates that range from −1 to 1 in all dimensions. (*1*) Every potential connection in the substrate is queried to determine its presence and weight; the *dark directed lines* in the substrate depicted 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, CPPNs can produce regular patterns of connections in space

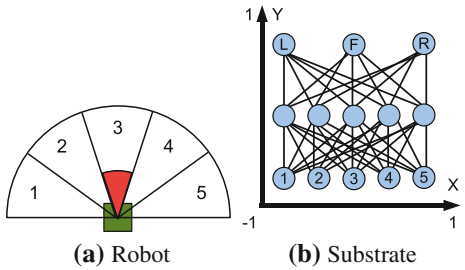


Fig. 4 Substrate configuration. An autonomous robot (**a**) is equipped with five sensors, spanning a 180° arc in front of it and labeled 1 through 5 from *left* to *right*. The substrate that controls the robot (**b**) is arranged such that the placement of inputs in the ANN corresponds to the physical locations of the sensors on the robot (for example, the leftmost sensor corresponds to the leftmost input). Similarly, the outputs of the network are related to their effects on the agent and correspond to the sensors (for example, the left turn output is on the left side of the network and above the leftmost sensor input). Such placement allows the CPPN to generate connectivity patterns easily that respect the geometry of the problem, such as left-right symmetry

that they exist on the robot (Fig. 4). Outputs for moving left or right can also be placed in the same order, implying a relationship between the sensors and effectors. In this way, knowledge about the problem geometry can be injected into the search and HyperNEAT can exploit the regularities (for example, 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. This capability is important for multiagent learning because it provides a formalism for producing policies (that is, the output of the CPPN) as a function of geometry (that is, the inputs to the CPPN). The evolutionary algorithm in HyperNEAT is the same as NEAT except that it evolves CPPNs that encode ANNs instead of evolving the ANNs directly. As the next section explains, not only can such an approach produce a single network but it can also produce a set of networks that are each generated as a function of their location in space.

3 Approach: multiagent HyperNEAT

Recall that the *policy geometry* of a team is the relationship between the canonical positions (physical or conceptual) of agents and their behavioral policies. Multiagent HyperNEAT is based on the idea that policy geometry is the right level of description for a team because it can be encoded naturally as a pattern, thereby describing the relationship of policies to each other. To understand how the policy geometry of a team can be encoded, it helps to begin by considering *homogeneous teams*, which in effect express a trivial policy geometry in which the same policy is uniformly distributed throughout the team at all positions.

Thus this section begins by exploring how teams of purely homogeneous agents can be evolved with an indirect encoding, and then transitions to the method for evolving heterogeneous teams that are represented by a single genome in HyperNEAT.

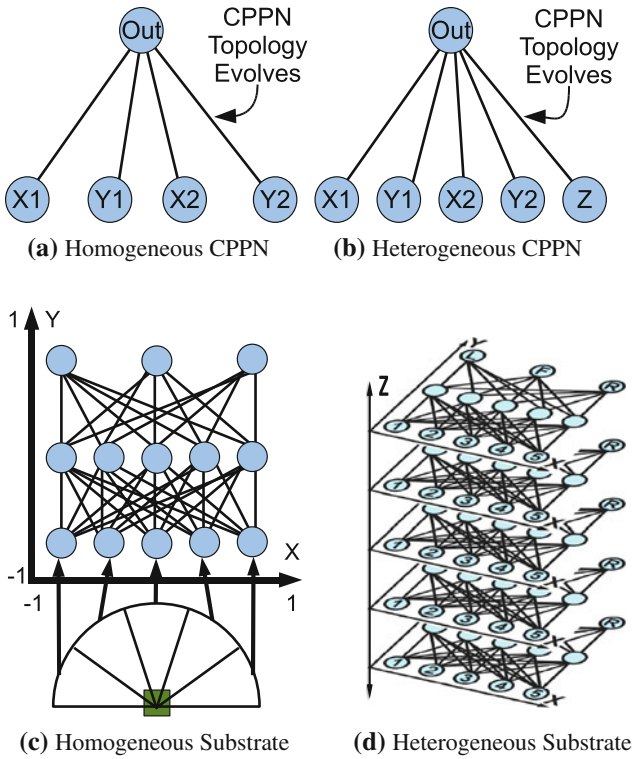
3.1 Pure homogeneous teams

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 (Fig. 5a) queries the substrate shown in Fig. 5c, which has five inputs, five hidden nodes, and three output nodes, to determine its connection weights. This substrate is designed to correlate sensors to corresponding outputs geometrically (for example, seeing something on the left and turning left). Thus the CPPN can exploit the geometry of the agent [96] when generating the ANN controller. However, the agents themselves have exactly the same policy no matter where they are positioned. Thus while each agent is informed by geometry, their policies cannot differentiate genetically.

3.2 Teams on the continuum of heterogeneity

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? Indirect encodings such as HyperNEAT are naturally suited to capturing such patterns by encoding the policy geometry of the team as a pattern. The remainder of this section discusses the method by which HyperNEAT can encode such teams.

The main idea is that the CPPN is able to create a pattern based on *both* the agent's internal geometry (x and y) *and* its position on the team (z) by incorporating an additional input (Fig. 5b, d). The CPPN can thus emphasize connections from z for increasing heterogeneity or minimize them to produce greater homogeneity. Furthermore, because z is a spatial dimension, the CPPN can literally generate policies based on their positions on the team. Note that because z is a single dimension, the policy geometry of this team (and those in this paper) is on a one-dimensional line. However, in principle, more inputs could be added, allowing two- or more dimensional policy geometry to be learned as well.



(a) Homogeneous CPPN **(b)** Heterogeneous CPPN

(c) Homogeneous Substrate **(d)** Heterogeneous Substrate

Fig. 5 Multiagent HyperNEAT encoding. The CPPNs and substrates in two approaches to encoding multiple agents with HyperNEAT are shown. 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 to produce a neural network is shown in **c**. In contrast, the CPPN in **b** encodes heterogeneous teams by sampling the heterogeneous substrate (**d**), which is made up of the single substrate (**c**) copied a number of times along the z-axis. Each discrete value of z corresponds to a new set of x and y coordinates that contain the controller for a single agent. By creating patterns across z, multiagent HyperNEAT can, in effect, exploit the policy geometry of the team. Note that CPPNs depicted in **a** and **b** increase in complexity over evolution through the NEAT algorithm

Thus each agent is assigned a z-coordinate based on its relationship to the other agents (for example, starting location) and a CPPN determines the weight of every connection within the agent's ANN by querying the connections using the five-dimensional CPPN as shown in Fig. 5b. The process for querying the networks for a team of agents is formalized in Algorithm 1.

For each agent a
  For each connection c in a
    output = CPPN(c.source.x,c.source.y,c.target.x,c.target.y,a.z)
    If $|\text{output}| > \text{threshold}$
      $\text{c.weight} = \text{Sign}(\text{output})\frac{|\text{output}|-\text{threshold}}{1-\text{threshold}}$

Algorithm 1: Querying a Multiagent Substrate (for example, Fig. 5d). The main idea is that if the CPPN output is above a threshold, then the connection is created and assigned a weight based on the CPPN output.

The heterogeneous substrate (Fig. 5d) formalizes the idea of encoding a team as a pattern of policies. This capability is powerful because generating each agent with the same CPPN means they can share tactics and policies while still exhibiting variation across the policy geometry. 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 encoding. The complete multiagent HyperNEAT algorithm is enumerated in Algorithm 2.

1. Set up the substrate to contain the necessary number of agents.
2. Initialize a population of minimal CPPNs with random weights that correspond to the chosen substrate configuration.
3. Repeat until a solution is found or the maximum number of generations are reached:
   (a) For each CPPN in the population:
      i. Query its CPPN for the weight of each connection in the substrate within each agent's ANN (Algorithm 1 and Fig. 3).
      ii. Assign the generated ANNs to the appropriate agents and run the team in the task domain to ascertain fitness.
   (b) Reproduce the CPPNs according to the NEAT method to create the next generation's population.

Algorithm 2: Multiagent HyperNEAT

Importantly, the complexity of the CPPN is independent of the number of agents in the substrate, which is a benefit of indirect encoding. Therefore, in principle, teams with a high number of agents can be trained without the additional cost that would incur to traditional methods. Another key property of the heterogeneous substrate is that if a new network is added to the substrate at an intermediate location, its policy can theoretically be interpolated from the policy geometry embodied in the CPPN. Thus, as the next section describes, it becomes possible to scale teams without further training by interpolating new roles.

3.3 Scaling after learning

As discussed in Sect. 2.1.1, there are typically no rules or principles to determine how *additional* agents could be added after learning has taken place in traditional methods. However, in real world tasks, it would be most convenient if the number of possible agents is unbounded and independent of the number initially trained. Whether in search and rescue operations or futuristic nanotechnology procedures, the potential utility of deploying hundreds of agents should be achievable through multiagent learning. Furthermore, because agents may break down or additional ones may become available, ideally the size of a learned team should be dynamically adjustable *after* deployment. While in the homogeneous case scaling is simply accomplished by adding or subtracting agents with the same control policy, scaling heterogeneous teams is in principle significantly more complicated.

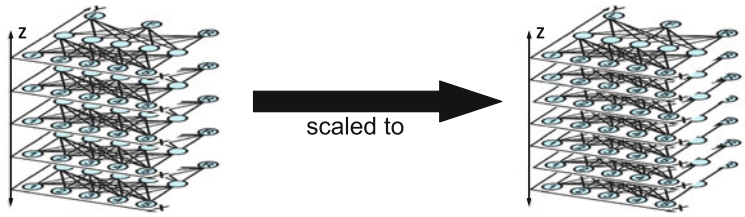
Recall the soccer team in Fig. 1, which includes eleven agents with assigned roles. How can additional agents be added to such a team, for example, between the midfielders and the forwards? Intuitively, the implicit policy geometry suggests that these new agents should interpolate between the policies of the surrounding agents, that is, they should be relatively offensive, but not as offensive as the players in front of them. Traditional techniques have no way to exploit this policy geometry because they treat each agent independently and would thus require retraining to assign roles to the new agents. However, because teams in multiagent HyperNEAT are represented by the CPPN as a *pattern of policies* rather than as individual agents, the CPPN effectively encodes an infinite number of heterogeneous agent policies that can be sampled as needed without the need for additional learning. Thus if more agents are required, the substrate can be updated to encompass the new agents and resampled to assign policies to them without further evolving the CPPN.

In fact, the heterogeneous substrate in Fig. 5d accommodates additional agents by simply redistributing their controllers on the z-axis so that they are uniformly spaced in accordance with their new number (Fig. 6).

Note that these steps can be taken *after* learning is completed and the new agent policies will be automatically interpolated based on the policy geometry by simply requerying the CPPN. There is no limit to the size to which such a substrate can be scaled in this way. Thus, through this approach, a new form of heuristic for post-deployment scaling is introduced.

3.4 Seeding

Seeding each agent of a multiagent team with the behavior of a single pre-trained agent exploits the fact that effective teams often lie close to pure homogeneity on the continuum of heterogeneity. That is, most teams include agents



(a) 5 Agent Team **(b)** 7 Agent Team

Fig. 6 Heterogeneous scaling. Because multiagent HyperNEAT represents teams as a pattern of policies, it is possible to interpolate new policies in the policy geometry for additional agents by sampling new points on the substrate. The original substrate (**a**) is scaled by inserting new two-dimensional substrate slices along the z-axis (**b**)

that share a number of skills and policies. For example, all members of a soccer team know how to kick, pass, and dribble a ball. Thus it would be inefficient if each agent had to learn these basic skills separately. In this way, training a single agent to master the core skill set is not only computationally more efficient (that is, only one agent needs to be simulated), but generally leads to more efficient learning of team-wide strategies. While most multiagent learning methods can seed teams or bootstrap learning intelligently through techniques such as layered learning [45, 98] (Sect. 2.1.1), because they are indifferent to the team policy geometry, they cannot subsequently vary the seed policy intelligently to coincide with the policy geometry.

In contrast, multiagent HyperNEAT creates teams as a function of their policy geometry and can alter the seed policy by gradually shifting it away from pure homogeneity along the continuum of heterogeneity. The challenge for multiagent HyperNEAT is to extend a strong single policy to represent the policies of an entire team. As with scaling, pure homogeneous teams are trivial to seed because such teams only require one controller that is copied to each member of the team; thus seeding the team requires only that evolution is started with a population of controllers derived from the seed. However, the same method does not work with heterogeneous teams because a CPPN that represents a single agent does not automatically represent a whole team. To address this problem, the remainder of this section describes how a CPPN that represents a single agent's controller can be slightly altered to encode the controllers of every agent on a team.

Recall that a CPPN that represents a single agent takes the four inputs x_1, y_1, x_2, and y_2 (Fig. 5a). This CPPN represents a two-dimensional connectivity pattern, whereas the stacked agent substrate is three-dimensional. Therefore, a new z input is added to the network, although with no connections to the existing network (Fig. 7). Only one z input is necessary because each agent exists at an infinitesimal point on the z-axis. In this way, the previously single-agent CPPN can now be queried for the policies of a *team* of agents. However, because the only factor that differentiates the agents, z, is not connected to the network,

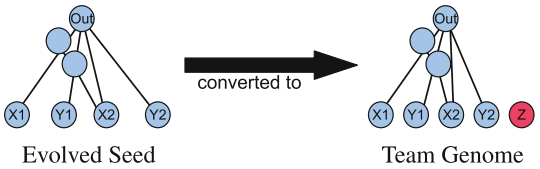
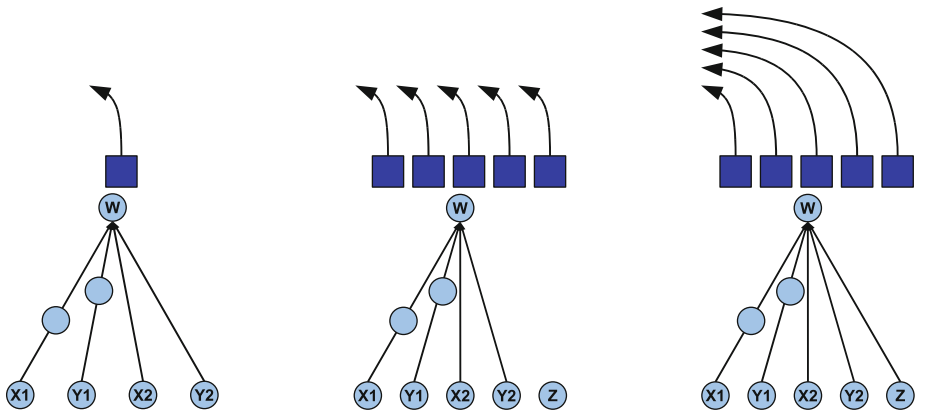


Fig. 7 Heterogeneous seeding. From a CPPN that generates a successful single agent, multiagent HyperNEAT can generate a team of agents. To allow the CPPN to differentiate between team members, the CPPN at left is modified by giving it a new z input that determines which agent is being sampled. This method preserves the original seed pattern, but allows multiagent HyperNEAT to create a pattern of policies relevant to the team's policy geometry, which varies along z

the team is initially homogeneous. However, once z is connected to the network by mutation, the CPPN can create variations of the seed policy based on the policy geometry along z. Figure 8 gives an example of how this idea works.

3.5 Applicability of approach

The multiagent HyperNEAT approach is a heuristic that can be exploited to train teams of heterogeneous agents. Like all heuristics, it is not ideal for all conceivable cases. For example, teams in which the agents do not ideally exhibit a clear or easily definable relationship, such as the relative aggressiveness of soccer players mentioned earlier, would be difficult to represent in this manner. However, as noted earlier, it is difficult to think of any real teams in which the members do not share some meaningful information even if it is as simple as how to move towards a target. Additionally, for teams where homogeneous behaviors are preferable a priori, multiagent HyperNEAT would be inefficient; although homogeneous teams can easily be represented by multiagent HyperNEAT (by ignoring the z input), the search process would continue to also search the super-set of heterogeneous solutions. In any case, it is incumbent upon the researcher to provide a substrate geometry that sufficiently encompasses these relationships in a manner that can be exploited by multiagent HyperNEAT. It is at this stage that domain knowledge can play a key role. For example, if a team must perform a symmetric task, ensuring that the agents are laid



(a) Single-Agent Behavior **(b)** Initial Seeded Team **(c)** Mutated Seeded Team

Fig. 8 Seeding example. A single agent (depicted as a *square* above its CPPN) learns the behavior of *turning left* (**a**), which is represented by its CPPN. This behavior can be used as a seed for a team of agents by simply adding a z input to the CPPN that represents the single agent's behavior. However, such a team is initially homogeneous because the z-input is initially unconnected (**b**). Through mutations the z input can become connected, allowing the CPPN to vary the initial seed policy among the agents on the team (**c**)

out symmetrically along z-coordinates and across the origin can be vital to solving the task. Additionally, while the first domain in this paper exploits the initial positioning of agents to define a policy geometry, there may exist other approaches that can also be effective. For example, policies could be assigned according to agent maximum velocities (for example, if agents are physically heterogeneous) instead of starting positions.

The next section describes a multiagent learning experiment designed to test the overall multiagent HyperNEAT approach and compare it to an existing multiagent learning method.

4 Predator–prey experiment

The aim of this experiment is to establish the advantage of representing a team as a pattern of policies and to demonstrate the novel capability to scale (with and without further training) that results. While traditional learning techniques offer potential solutions to multiagent problems [67, 97, 121], such approaches do not exploit fundamental regularities that govern the distribution of policies on a team, or the fact that such policies tend to overlap partially and can thus benefit from a representation that does not need to relearn shared information. To test that exploiting such regularities is beneficial, teams trained with multiagent HyperNEAT are compared against teams trained with Sarsa(λ) [104], a traditional reinforcement learning technique that was chosen because it has been applied to multiagent learning [53, 101], in a multiagent predator–prey problem. There are many domains (such as Robocup soccer [100]) that can benchmark an approach. However they are typically not designed with scalability in mind; Adding or removing agents can trivialize or overly complicate the problem being solved. With that in mind, cooperative multiagent predator–prey is a good platform to test this idea because the task is challenging yet easy to understand. Both methods are equally challenged because this domain, like many multiagent domains, is partially observable (that is, non-Markovian) and both temporal difference methods and the feed-forward neural networks that control multiagent HyperNEAT's agents (Fig. 5d) in this task typically could only provide performance guarantees in Markovian domains. However, both methods have been successfully applied in particular to multiagent predator–prey problems in the past [25, 49, 54] (although at different team sizes), so this task will provide a good benchmark.

The version of predator–prey in this paper is similar to that described when multiagent HyperNEAT was first introduced [25]: Agents *cannot see* their teammates nor can they communicate with each other. Also, 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. Therefore, 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 multiagent HyperNEAT creates all the agent ANNs from the *same* CPPN, it has the potential to balance these delicate ingredients. An important difference between the predator–prey domain in this investigation and in others [49, 50, 54] is that the environment is neither a fixed size nor toroidal, allowing an arbitrary number of predators and prey to fit in the world, and facilitating the later scaling experiments. The domain also differs from the original multiagent HyperNEAT predator–prey domain in that the agents are limited to a set of discrete actions, which makes

the domain more amenable to reinforcement learning techniques such as Sarsa(λ). Also unlike previous scaling experiments with multiagent HyperNEAT [25], the domain difficultly (that is, number of prey) scales with the number of predators, providing an additional challenge for scaled teams.

4.1 Predators and prey

The goal of the predators is to capture (that is, intercept) the prey agents by positioning themselves so that a prey is within 3 U of the predator. The predators are equipped with five rangefinder sensors evenly spanning a 180° arc that detect prey within 50 U. Predators *cannot* sense each other. Thus, the state of each agent is the five continuous floating-point values (0.0–1.0) returned by the sensors, representing the distance of the closest prey in the corresponding arc (as in Fig. 4a). In the case of the HyperNEAT-controlled agents, these five values correspond to the five inputs on the substrate (Fig. 4b) that is queried by the CPPN to generate the agent's controller. In contrast, to facilitate reinforcement learning, Sarsa(λ) encodes these values with Sutton's tile coding algorithm [105], with eight tilings and a tile width of 0.2 to create the state representation. Tile coding is a coarse coding method used to increase generalization and encode large or continuous state spaces compactly for reinforcement learning, and has previously been applied to MARL [101, 115]. The state space is partitioned into several tilings, which are further broken into tiles, such that each tile representing a discrete feature. The number of tilings and tile width were chosen to encourage generalization, and because they produced the best results in preliminary experiments. For a complete description of tile coding see Sutton [103].

At each discrete moment of time, a predator can turn 90° left or right or move 1 U forward. HyperNEAT-controlled agents have three outputs in their neural networks (Fig. 4b), one for each such action, and perform the action with the highest activation after the network is activated. Reinforcement learning agents independently maintain and update their own Q-functions and select actions with an ϵ-greedy approach during training, where $\epsilon = 0.01$, and pure-greedy action selection during testing. For both methods, ties default to moving forward.

Prey agents are programmed to maintain their current location until they are threatened; if there is a predator within 5 U the prey moves in the direction exactly opposite to the direction of the closest predator; thus the prey are not restricted to discrete actions and can adjust their heading to any angle (for example, a prey can move diagonally to escape a predator). Prey move at twice the maximum speed of predator agents. That way, it is impossible for a single predator to catch a prey and the predators must work together to accomplish their goal.

The predator team starts each trial in a line, 4 U apart, facing the prey (Fig. 9). The environment the agents inhabit is physically unbounded, and each trial lasts 1,000 time steps. On Sarsa(λ) teams, each agent receives the same reward r at each time step, where $r = \frac{p_c}{p_t} - 1, p_c$ is the number of prey captured, and p_t is the total number of prey, until either the time limit expires or all prey have been captured. This reward scheme follows precedent in prior applications of MARL that reward all agents collectively for team success [101]. Trials were also attempted with individual rewards, but performance was markedly worse, further supporting the chosen scheme. Because they are evolved, HyperNEAT teams do not receive rewards over the trial; instead they are given a fitness value of $\left(1,000\frac{p_c}{p_t}\right) + (500 - t)$, where t is the time taken to capture all the prey or 500 if not all prey were captured. Both measures incentivize the predators to capture as many prey as possible, as quickly as possible.

The major challenge for the predators is to coordinate despite their inability to see one another or communicate. This restriction encourages establishing a priori policies for cooperation because agents thus have little information to infer each others' current states. Such scenarios are not

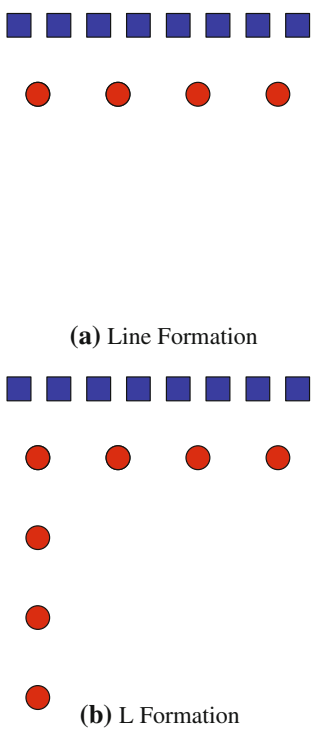


Fig. 9 Prey formations. The prey (*circles*) are arranged in either a line (**a**) or an L (**b**) formation, thereby testing both symmetric and asymmetric scenarios. This figure depicts both formations with eight predators (*squares*), although the number of predators and prey can change. The predators are always placed in an evenly-spaced line below the prey

uncommon. Military units often form plans, split up, and execute complicated maneuvers with little to no contact with each other [29].

4.2 Prey formations

Agent teams are trained on one of two formations: the line or the L (Fig. 9a, b, respectively), both of which presents a different challenge to the teams.

In the line, there are half as many prey as predators. The prey are positioned 10 U away from the predators vertically and spaced $\frac{4(2p_t-1)}{p_t}$ units apart horizontally, starting from the same x-coordinate as the predators, but shifted by half the normal spacing amount. Thus the prey are spread across the same distance as the predators, but with a slight space on the edges to encourage predators to approach the prey. An interesting property of the line regarding cooperation is that it is symmetric, so teams should in principle be able to develop symmetric strategies to capture the prey. Additionally, all the prey are seen by at least one predator at the start of the simulation so exploration is not required, that is, together the predators have complete information about the prey. However, because they cannot communicate, individual predators still have limited state information.

The L formation is a more complex test of coordination for the teams because it is asymmetric. Thus the agents have to learn specific roles for specific locations. Preliminary multiagent HyperNEAT experiments only tested symmetric prey formations, but were able to develop both symmetric and asymmetric strategies [25], suggesting that such blind coordination should be possible in principle. Also, the prey are not fully-observable from the start, that is, many of them cannot be seen by any predators when the simulation begins. Thus exploration is necessary to solve the task. Finally, the L has almost double the prey compared to the line formation, which makes the task more complex and time-consuming. The first half of the prey are placed exactly as in the line formation. The remaining prey are placed behind the left-most prey in a straight line back, 5 U apart. Note that unlike the line, in the L formation the maximum distance between a predator and a prey increases depending on the number of prey. Thus in sizes 128 and 256 of the L a predator cannot actually reach all the prey within the time limit, so these sizes will not be investigated. Also, size two of the L is identical to the line at size two, so it will also not best tested in these experiments.

4.3 Scaling

As technology progresses, the demand for larger, more complex multiagent teams increases. Thus, the ability to train scalable teams of agents is critical. In this paper, two types of scaling will be investigated: pre-training and post-training.

A multiagent learning algorithm should be able to handle training a large number of agents simultaneously. In this paper such scaling is referred to as *pre-training* scaling because the number of agents in the team is known before training is started. Thus this type of scaling tests the scalability of the learning algorithm itself. Such a property is important if large-scale, real world problems are to be solved with the method.

Post-training scaling, in contrast, is defined as changing the size of the team *after* training is already completed and *without further learning*. Such scaling is challenging because the policies of the new agents must be assigned automatically. However, this capability would benefit real-world multiagent applications such as unmanned aerial vehicle (UAV) or unmanned ground vehicle (UGV) swarms. For example, imagine that additional UGVs become available to an existing swarm of UGVs on a search and rescue operation. The ability to integrate these new agents into the team immediately could be critical. Many traditional multiagent learning approaches are not designed to facilitate such scaling because they represent agents as discrete entities. In contrast, because multiagent HyperNEAT represents a team of agents as a pattern of policies, it is possible to dynamically change the size of a team by sampling new points in this pattern (Sect. 3.3). This kind of *interpolated scaling* is a unique capability of HyperNEAT, making a direct post-training comparison between methods difficult. Therefore, because Sarsa(λ) lacks such a capability, its post-training scaling will be tested by duplicating the policies of existing agents such that when a team is scaled up by a factor of S, the first S agents will have the first agent's policy, the second S agents will have the second agent's policy, and so on.

Therefore, both methods will be tested in their ability to scale both pre and post-training with team sizes ranging from 2 to 1,024 agents, wherein each team size is double the last (that is, 2, 4, 8, and so forth). Teams will be trained on sizes of up to 256 agents for the line and 64 for the L and tested on up to 1,024 for the line and 64 for the L.

4.4 Seeding

Simulating multiagent teams is typically computationally expensive. Post-training scaling is one way to mitigate this issue. However, another is to take existing single-agent policies and build a team from them as a starting point for further learning. In this way, knowledge about fundamental or important skills and strategies can be injected into the search from the beginning. Such a capability is powerful because starting with fundamental skills means they do not

need to be discovered by the search algorithm. Also, it can be much less expensive to simulate and train a single agent to perform the required skills.

Seeding in multiagent problems has been used in both evolution [45] and Sarsa(λ) [110] with success. For multiagent HyperNEAT, Sect. 3.4 described how a team is formed from a single seed genome. For Sarsa(λ), the weights of a single agent are normalized and copied to all agents on the team. In both cases the single, seed agent was trained to chase prey as closely as possible (recall that a single predator cannot capture prey, so chasing is the best starting point possible). Solutions were found in both cases in under 5,000 evaluations and form the basis for seeding experiments.

Although the initial seeded team for both methods (that is, a homogeneous group of predators that can chase prey) is unlikely to be able to solve the problem directly, such a team should provide a good starting point for agents to differentiate and then solve the problem. The key difference between Sarsa(λ) and multiagent HyperNEAT in this respect is that multiagent HyperNEAT can discover a policy geometry with which to vary this base policy, whereas Sarsa(λ) must independently learn how to change each agent's individual behavior to best suit the team's goal. To verify that this capability is important, multiagent HyperNEAT and Sarsa(λ) are also tested with seeded policies.

4.5 Experimental parameters

Because HyperNEAT differs from original NEAT only in its set of activation functions, it uses the same parameters [90]. Experiments were run with a modified version of the public domain SharpNEAT package [38]. 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 CPPN 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 [90, 92, 94]. They were found to be robust to moderate variation through preliminary experimentation.

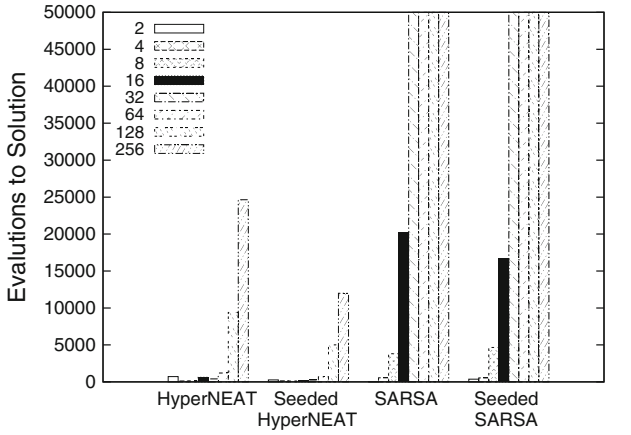
In the Sarsa(λ) runs, the standard Sarsa(λ) update rule is used [104] with $\lambda = 0.9, \epsilon = 0.01$, and $\alpha = 0.05$. In addition, the implementation uses Sutton's tile coding software [105] with five variables (the sensor readings), eight tilings (different discretizations of the state space), and a tile width of 0.2 (size of the tiles in the discretizations). These values were found to produce the best results through preliminary experimentation.

5 Results

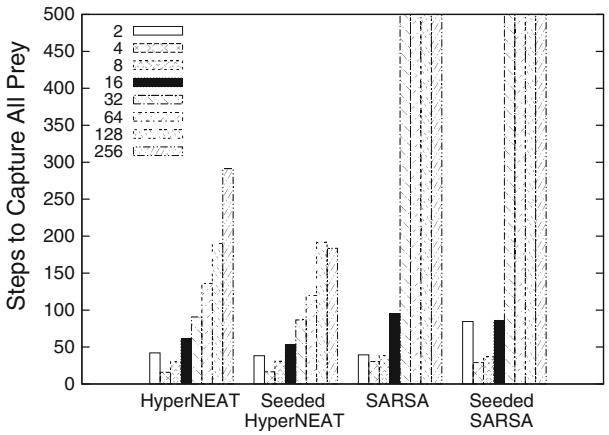
To test the pre-training ability of each method to scale, teams were trained on sizes 2, 4, 8, 16, 32, 64, 128, and 256 for the line and 4, 8, 16, 32, and 64 for the L. The main question is whether the methods can continue to find effective solutions as team size (and also the number of prey) increases. Note that because the states of the agents are egocentric and the agents do not see each other, the state-space does not necessarily grow for *individual* agents as in other multiagent problems. However, as more prey are added, the possibility of more sensors being simultaneously activated at different values does increase, meaning that an agent is more likely to encounter a larger number of more varied states as the number of prey increase. Additionally, as more agents are added, the potential for conflicts among agent policies increases. Figures 10 and 11 show how each method scales with pre-training (that is, the teams were trained at the desired team size) on the line and L formations, respectively. In both figures, the first graph shows the average number of evaluations until the first solution was found and the second graph shows the average number of timesteps during simulation until all the prey are captured for the best solution of each run. In both cases lower values are better and if a bar is at the maximum value (50,000 evaluations or 500 timesteps), it means that no teams were able to solve the problem at that size for that method.

For the line formation, the most striking result is that while HyperNEAT is consistently able to find solutions for teams of up to 256 agents, Sarsa(λ) stops being able to find solutions above only 16 agents. In fact, each time the number of predators and prey doubles, the number of evaluations Sarsa(λ) requires to solve the problem increases by an order of magnitude. The results are similar in the L: multiagent HyperNEAT finds solutions for all tested sizes, but Sarsa(λ) can only find solutions for size four. Therefore, multiagent HyperNEAT, with the ability to encode and learn the policies of an entire team and the relationships among them simultaneously, is better able to deal with the conflicts that arise in a large multiagent domain.

For the team sizes that both methods could solve, multiagent HyperNEAT was only significantly faster in terms of capture speed at eight and 16 on the line formation and four for the L ($p < 0.01$ according to Mann-Whitley U test for all), implying that at smaller sizes both methods are able to refine existing solutions once they are found, but that actually finding an initial solution with separately-represented agents is a more difficult problem. Furthermore, for a two-predator team, the smallest size, Sarsa(λ) finds solutions significantly faster ($p < 0.01$) than multiagent HyperNEAT, implying that reinforcement learning

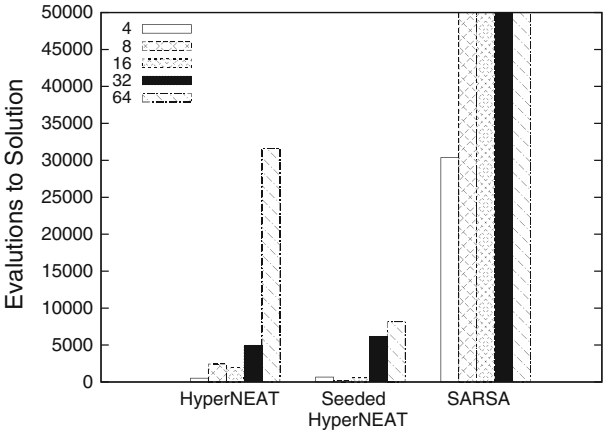


(a) Time to Find Solution

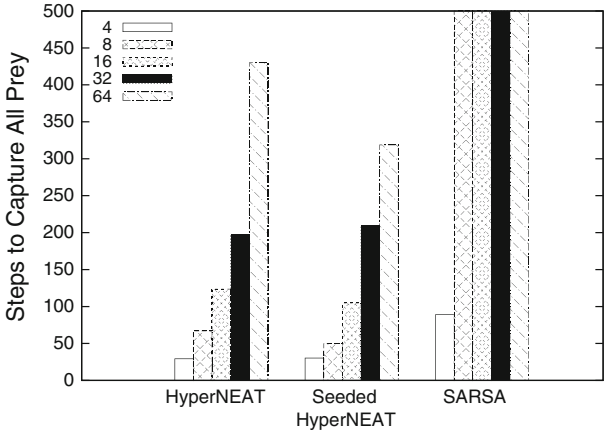


(b) Time to Capture Prey

Fig. 10 Line pre-training scaling. The performance of each method for pre-training scaling on the line formation is shown. In almost all cases multiagent HyperNEAT teams are able to find better solutions more quickly than Sarsa(λ). After 16 agents, Sarsa(λ) can no longer solve the task, while HyperNEAT is still able to solve it up to 256 agents. Results are averaged over ten runs



(a) Time to Find Solution



(b) Time to Capture Prey

Fig. 11 L pre-training scaling. The L formation was more difficult than the line due to the exploration necessary to solve the problem. However, multiagent HyperNEAT was able to find solutions consistently for all tested sizes. Sarsa(λ) could only find solutions at the smallest size of four, and seeded Sarsa(λ) was unable to find any solutions. Note that because the distance between the furthest prey and the predators doubles each time the size increases, it is also reasonable for the time to capture the prey to double as well. Results again are averaged over ten runs

may be more efficient for small teams and demonstrating that team size is the relevant variable in this comparison.

Note that the fact that HyperNEAT and Sarsa(λ) are comparable up to about eight agents is an important validation that the Sarsa(λ) implementation works. Of course, an important concern in comparisons between different approaches is that all are implemented properly; the competitive results at the smaller scales provide evidence that the disparity at higher scales is not only implementation-dependent.

Seeding generally improved multiagent HyperNEAT's performance, although there are some instances where both seeded and unseeded teams were able to find an optimal solution or where seeding hurt performance. With Sarsa(λ), seeding was only beneficial at size 16 on the line, indicating that just having a good seed policy may not be enough; the way in which these policies are manipulated during learning is also critical.

Post-training scaling is more challenging: More agents are added to the team without further training, which means the policies of the new agents must somehow be intelligently derived from the old. Because HyperNEAT encodes the team as a pattern, it can create the policies of the new agents by querying the CPPN at an intermediate location, thereby creating an interpolation of the existing policies. Multiagent Sarsa(λ) has no such mechanism to automatically generate new policies. In this sense it is difficult to compare them when Sarsa(λ) lacks an analogous capability. However, it is still an important question whether interpolating new policies gains any real

advantage over simply duplicating the range of existing policies to make a larger team. Therefore, to validate the contribution of role interpolation, scaled Sarsa(λ) teams contain multiple duplicates of the agents in the unscaled version, as described previously. The post-training results were obtained by testing all saved champions (that is, teams that scored better than all previous teams during the run) on all sizes to determine the best scalers for each run. This method of testing generalization follows Gruau et al. [39] and is designed to compare the best overall individuals. In addition to scaling up, scaling down is also tested. Because the number of opportunities to scale up or down depends on the training size being tested, teams are tested and evaluated separately on scaling up and down, but they are displayed together.

Figures 12 and 13 show post-scaling results on the line and L formations, respectively. Each graph shows the number of runs in which the best scaler from each run can scale to a target size out of ten runs, starting from teams trained at different sizes. If a method was not able to find a solution at a particular pre-trained team size then that size is not included as a starting point for post-scaling training. For the line, Sarsa(λ) was only able to scale at all from the smallest two-agent size and only some runs were able to find scalable solutions. In contrast, multiagent HyperNEAT found scalable solutions from all starting sizes (except in the case of size 64 on the L, which is the largest L size possible within the time limit). For small team sizes, multiagent HyperNEAT was consistently able to find solutions that can scale at least up to the next team size, although in most cases teams scaled several sizes up and down. The results were similar on the L, with the exception of size 64, from which it is not possible to scale within the time limit. Scaling up above 128 agents on the line proved to be difficult for multiagent HyperNEAT, which typically only found solutions that could scale to such sizes one to three times out of ten. However, *seeded* multiagent HyperNEAT was able to rarely find solutions that scaled up to 1,024 agents, a team size that no method could solve when trained to solve it. Thus policy interpolation produces a significant new potential to scale already-trained teams.

5.1 Post-training scaling with further learning

While it is interesting from a research perspective that multiagent HyperNEAT can scale to different team sizes without further learning, as a practical matter, there is no reason that the teams cannot continue learning at the new size to further optimize or correct minor imperfections in the scaled policy. This approach is similar to incremental evolution and shaping techniques that have been applied in other multiagent scenarios [16, 65, 81]. In fact, while many policies generated by multiagent HyperNEAT are able to scale to different team sizes, those that do not scale perfectly still generally display an appropriate (though not perfect) strategy based on policy geometry, indicating that the scaled policy is close to a correct solution but may exhibit some artifacts in the policy geometry that were not sampled during training. Previous HyperNEAT research [96] showed that similar artifacts are present when scaling the sensors and effectors of a single agent, but that such artifacts are easily smoothed by additionally learning. To test that scaled teams trained with multiagent HyperNEAT can be further trained to correct imperfections, the three best scaling teams at size 64 are further trained at size 256 (to which no unseeded team trained on 64 agents could scale without further learning) on the line formation. Although Sarsa(λ) was not able to train a team of 64 agents, to facilitate some comparison, and to test whether this capability is unique to HyperNEAT, teams trained by Sarsa(λ) with four agents are scaled to eight agents and further trained.

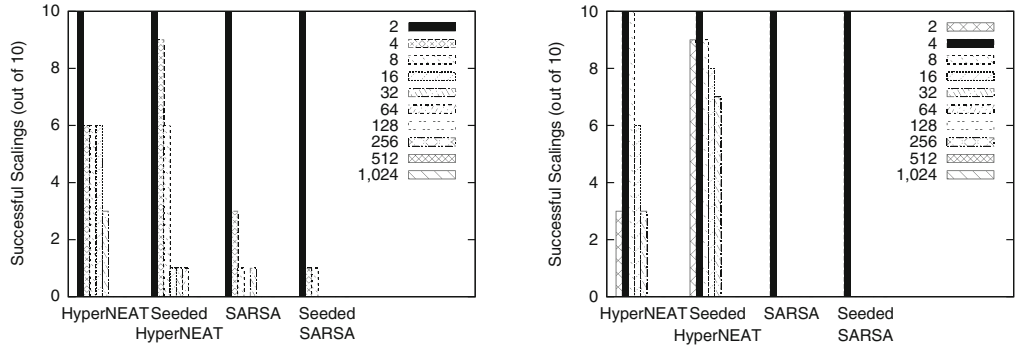
Multiagent HyperNEAT was able to find solutions that solved the new size in 1,220 evaluations on average, which is significantly faster than finding a solution from scratch on 256 agents (which takes on average 24,650 evaluations). In contrast, the number of evaluations for Sarsa(λ) to find a solution at the large size after scaling (average 4,764 evaluations) did not differ significantly from the number required to find a solution for eight agents from scratch (which takes on average 3,804 evaluations). Thus while it is possible for multiagent HyperNEAT to find solutions that scale without further learning, even those that do not scale perfectly still retain the information necessary to solve the problem at different scales; the solution simply must be tweaked through a small amount of additional learning to accommodate the new size. Additionally, the fact that Sarsa(λ) does not benefit from this bootstrapping approach implies that policy geometry plays a critical role in further training after scaling.

5.2 Typical behaviors

This section describes the typical behaviors produced by both learning methods. These qualitative results are important because they illuminate why the quantitative results make sense. Videos of these behaviors can be found online.[1]

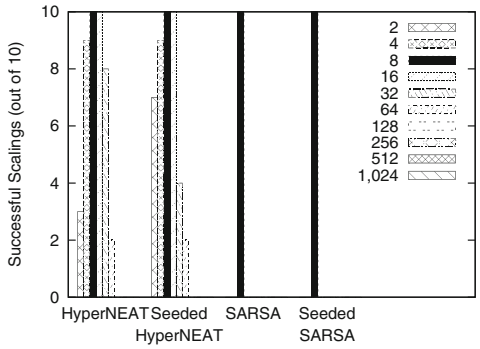
In the line, on the smallest team sizes (two, four, and eight), the typical strategies employed by multiagent HyperNEAT and Sarsa(λ) do not differ greatly. The first solution discovered at these sizes involves one or more agents moving behind the group of prey and pushing them towards the remaining agents. On sizes four and eight,

[1] http://eplex.cs.ucf.edu/demos/multiagentcompared.

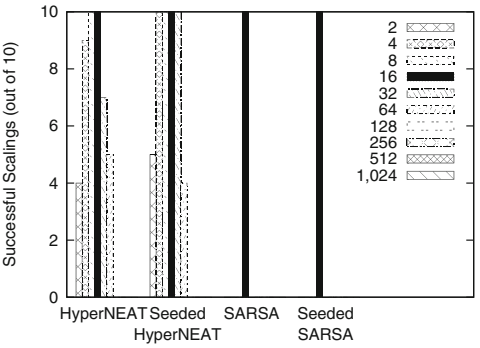


(a) 2 Agent Scaling

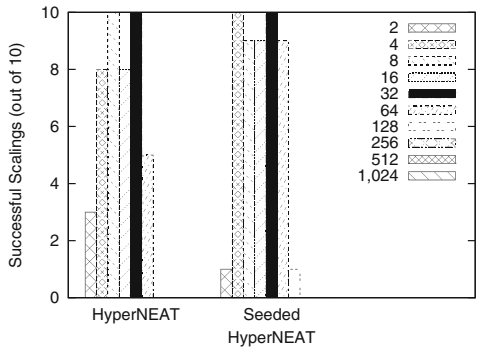
(b) 4 Agent Scaling



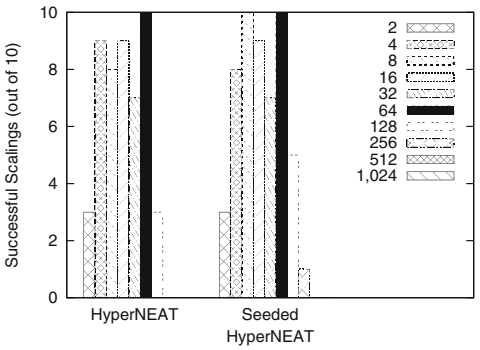
(c) 8 Agent Scaling



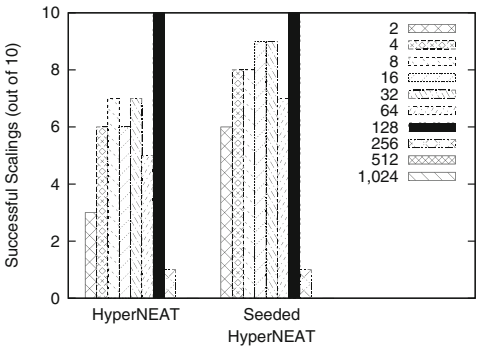
(d) 16 Agent Scaling



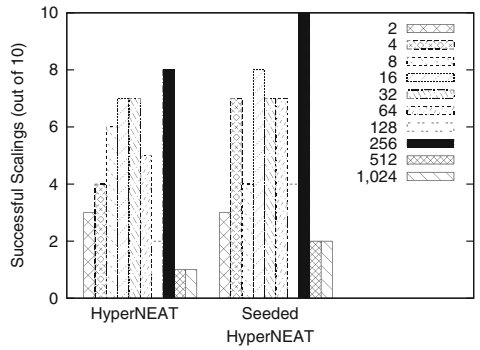
(e) 32 Agent Scaling



(f) 64 Agent Scaling



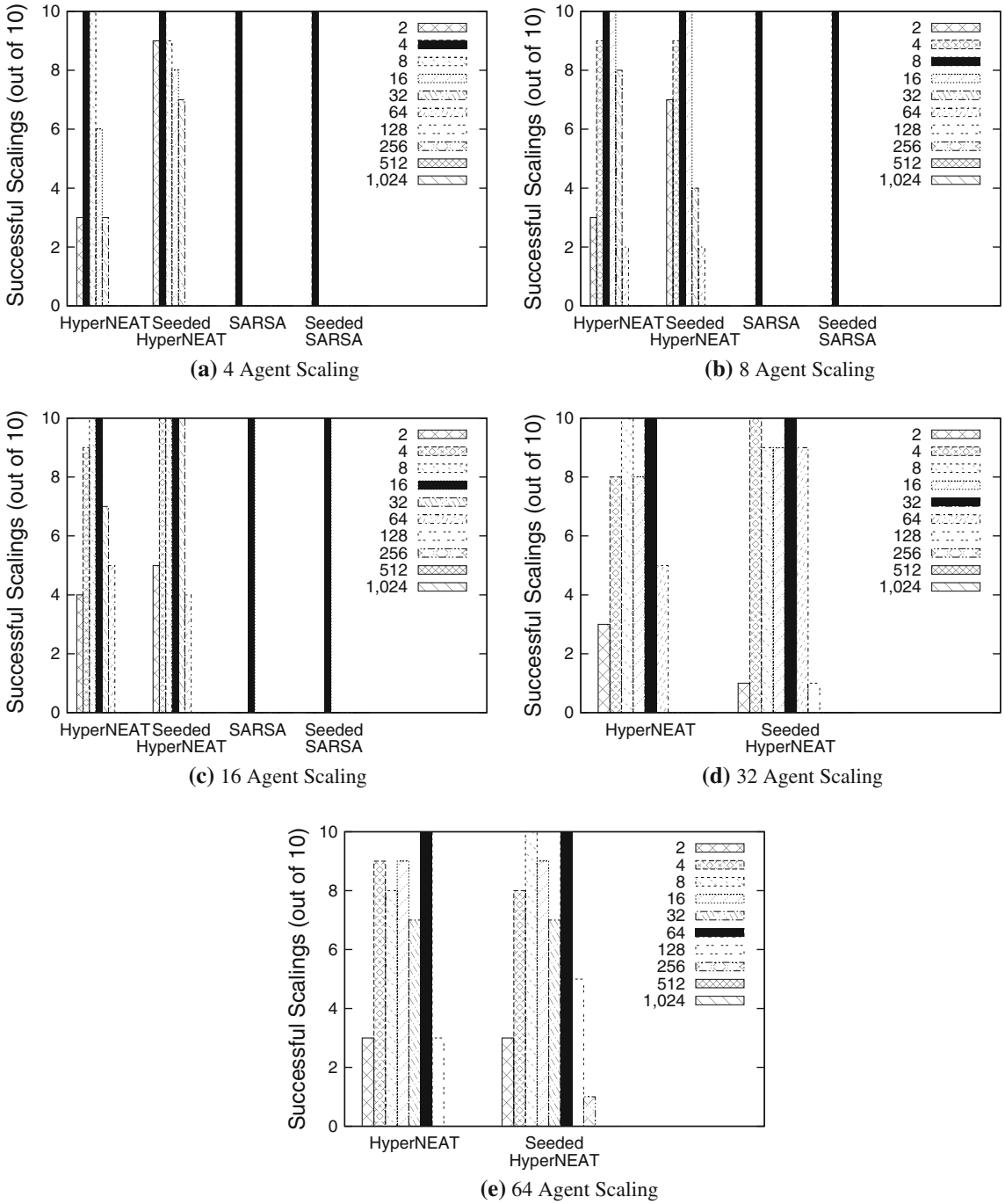
(g) 128 Agent Scaling



(h) 256 Agent Scaling

Fig. 12 Line post-training scaling (higher is better). The number of successful scalings (both *up* and *down*) out of ten runs for different team sizes is shown. For each *graph* the training size is *shaded black*. While HyperNEAT is able to find scalable solutions at all sizes without further learning, Sarsa(λ) can only scale from two-agent solutions (up to at most 16)

teams then transition into some or all of the predators surrounding and capturing the prey agents. At size 16, Sarsa(λ) continues to employ the *same* strategies (Fig. 14a–c), but because the prey line is longer and there are more agents to coordinate it takes much longer to find a solution and for that solution to capture the prey. In contrast, while multiagent HyperNEAT also first finds such solutions, it quickly discovers different strategies that divide the prey into two groups that the predators independently surround (Fig.14d–f), exploiting the symmetry of the team. Beyond size 16, Sarsa(λ) stops solving the problem because a more coordinated approach is needed.



(a) 4 Agent Scaling

(b) 8 Agent Scaling

(c) 16 Agent Scaling

(d) 32 Agent Scaling

(e) 64 Agent Scaling

Fig. 13 L post-training scaling (higher is better). Successful scalings (out of 10) are shown on these graphs for different team sizes. The training size from which teams are scaled is shown in *black*. HyperNEAT was able to find scalable solutions for all sizes that fit within the time limit, while Sarsa(λ) could not scale at all

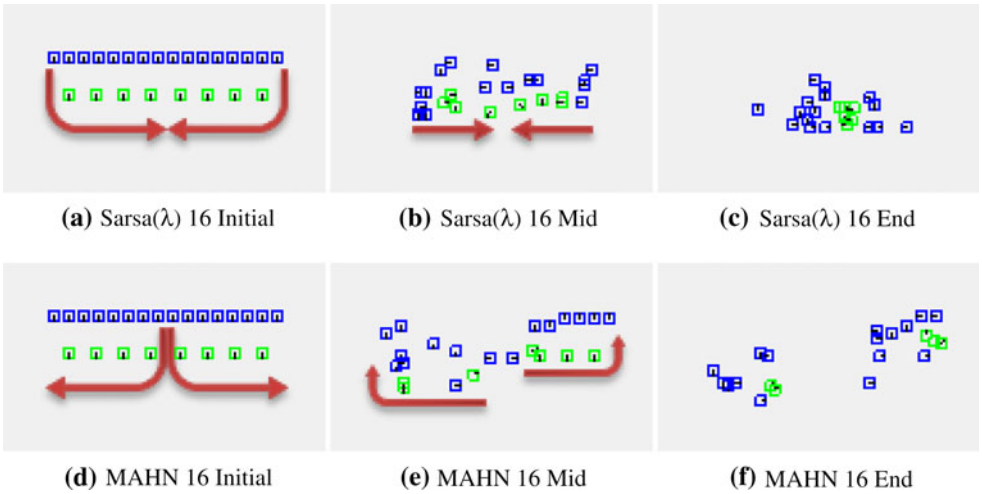


Fig. 14 Typical strategies for 16 predators. At size 16 predators (*blue squares* at the top of the pictures), Sarsa(λ)-trained teams typically try to surround the prey (*green squares* closer to the bottom) by employing the predators on the edges to move to behind the prey (**a**) and push them towards the center (**b**) where they can be captured (**c**). Multiagent HyperNEAT teams also learn this strategy, but eventually develop a more complex version wherein the predators divide the prey into two groups (**d**) and then surround (**e**) and capture them independently (**f**). The multiagent HyperNEAT result is more efficient because more prey are captured in parallel, and it is more scalable because eventually the size of the prey lines becomes too large to traverse within the time limit

The first solutions multiagent HyperNEAT finds at sizes 32 and 64 tend to be the same strategies as at 16, but at these sizes they are suboptimal and act as stepping stones to strategies that divide the prey into multiple groups that are captured independently (Fig. 15a–c). Multiagent HyperNEAT is able to find such strategies by repeating coordinate frames within the policy geometry.

Finally, at sizes 128 and 256 simple strategies are no longer viable due to the length of the line of prey, as seen by the fact that successful scaling to these sizes from smaller sizes drops off sharply in Fig. 12. Thus the first strategies that are found are those that split the prey into multiple groups. Some runs at these large sizes were also able to find an alternative strategy wherein every other predator does nothing and the remaining predators move forward, causing the prey to bounce between each pair, until each prey runs into one of the predators while trying to avoid the other (Fig. 15d–f). Such a strategy is simple and effective, but would be very difficult to learn if all agents were represented independently. Seeded and unseeded versions of the same method did not cause a significant qualitative difference in behavior; better behaviors were just found faster.

On the L, at size four, both methods used the same strategy of surrounding the prey to capture them. At size eight, Sarsa(λ) can no longer solve the problem, but multiagent HyperNEAT teams learn a strategy to deal with the vertical portion of the L similar to that of the large teams on the line, that is, the first and third agent move forward and capture the entire line by bouncing the agents between them. The horizontal portion of the line is captured by the remaining predators through a surround strategy. The remaining sizes (16, 32, and 64) capture the vertical part of the line by sending a group of agents down one side of the prey, pushing them slightly to the right, while another group charges straight at them. The combination of these two actions forces the prey into a compact ball that is pushed downward by the charging predators. When the first group of predators reach the end of the vertical portion of the L, they turn right to form a barrier between the charging predators and the ball of prey, and when the two meet, the entire ball is captured. The horizontal portion is again captured by a simple surrounding strategy. There is little incentive for the predators to improve upon the horizontal capture speed at these sizes because the time taken to capture the vertical portion of the prey dominates the problem. Nevertheless, the results on the L formation show that multiagent HyperNEAT can discover effective asymmetric strategies for capturing prey.

The few Sarsa(λ) solutions that could scale up post-training did not typically resemble the solutions they scaled from, although the only scalable solutions came from two-predator, one-prey experiments in which the overall policy is difficult to discern. In contrast, most post-training scaled multiagent HyperNEAT solutions did resemble the solutions from which they were scaled, but sometimes contain some inefficiencies such as a predator pushing the prey in the wrong direction initially. A key factor in determining whether a team will scale post-training is whether the team employs a strategy that is effective at a higher level. For

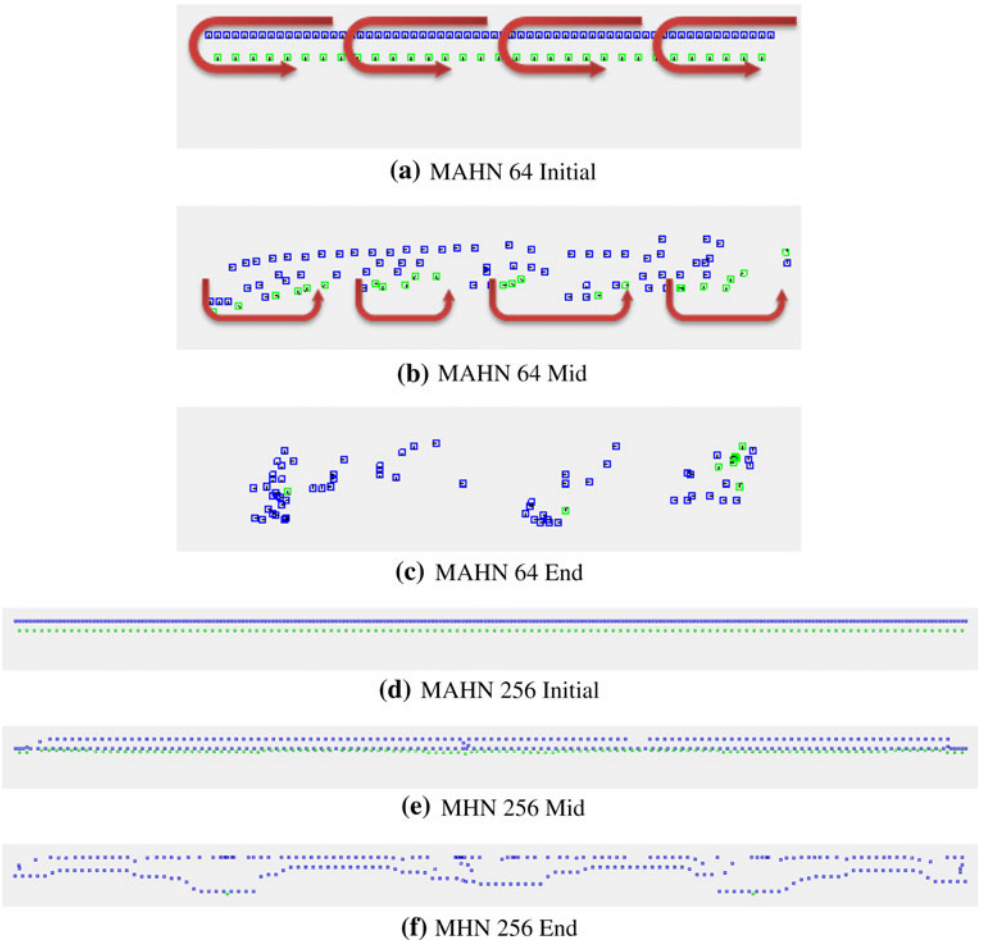


Fig. 15 Multiagent HyperNEAT Strategies for Large Teams. For larger teams such as the team of 64 predators in this figure, multiagent HyperNEAT typically continues the strategy of dividing the prey into groups (a), surrounding (b), and capturing them (c). The difference from smaller sizes is that they are divided into more and more groups so that larger numbers of prey can still be efficiently captured by the predators. However, at the very largest trained size of 256, multiagent HyperNEAT sometimes found a strategy wherein every other predator does nothing and the remaining predators move forward (d). This strategy causes the prey to bounce between the charging predators (e) until they are all eventually captured (f). Such a tactic is extremely efficient at this problem size and requires the strict cooperation of almost every agent to be successful

example, if a team trained at size 16 exhibits the strategy of positioning a single agent behind all the prey and pushing each of them towards the predators, the strategy has less of a chance of transferring to larger teams. However if the team instead learned how to divide the prey and capture them as groups at size 16, its chances of scaling are much greater.

6 Discussion

In the predator–prey domain, the result that heterogeneous teams trained with multiagent HyperNEAT significantly outperform teams trained with Sarsa(λ) in both training and scaling demonstrates the importance of exploiting team geometry in multiagent learning. Whereas teams trained with Sarsa(λ) were unable to solve problems with over 16 agents, pre-trained multiagent HyperNEAT teams solved up to 256 agents. In this way, 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. Multiagent HyperNEAT allows team behavior to be represented as variation on a theme encoded in a *single genome*, which means that key skills need not be rediscovered for separate agents, overcoming the problem of reinvention. 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.

The number of evaluations taken by multiagent HyperNEAT to find a solution increases greatly at 128 agents. This change reflects a fundamental discontinuity in the policies required to solve the problem at smaller and larger sizes. At smaller sizes, a common early solution is for a small subset of agents to capture all the prey while the others are not involved. While such a solution is suboptimal, it is a solution nonetheless and is a stepping stone to more efficient solutions that make use of the whole team. However, if learning is first to succeed only with a subset of the team, a subset of predators must be able to visit each of the prey. Yet when there are 128 predators, the line of prey becomes longer than the distance a single predator can traverse in the time limit. Thus, at large sizes, such suboptimal policies represent local optima that cannot lead to efficient solutions and the only viable strategies are the more complex coordinated ones that employ more

predators and take longer to find. While HyperNEAT can find these with effort, the required coordination is too much for Sarsa(λ). There do exist specific approaches for encouraging coordination, such as allowing agents to make inferences about the behaviors of their team mates [51]. However, such approaches add complexity to the individual agent policies and become intractable when agents must consider large numbers of cooperating agents.

Seeding was generally beneficial to both methods after size 8. While performance was generally unaffected or even hurt at smaller team sizes, where simple solutions were easily found and some of the seeded behavior may need to be unlearned, it generally improved at the larger sizes. This capability captures the idea that real-life teams (for example, in soccer) often share a critical basic skill set that can be learned faster by an individual agent than by an entire team, thereby exploiting the team's position on the continuum of heterogeneity. While HyperNEAT naturally 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.

Multiagent HyperNEAT was able to create teams of agents that could perform well at sizes orders of magnitude larger than the training size without further training. Such scaling is prohibitive for Sarsa(λ) and other traditional heterogeneous multiagent learning techniques, which do not implement interpolated scaling, yet multiagent HyperNEAT accomplishes it by representing the teams as patterns rather than as individual agents. Even though these patterns are sparsely sampled during training (that is, with orders of magnitude fewer points) the scaled teams exhibit smooth interpolations between agent roles. A particularly interesting result is that while multiagent HyperNEAT could not solve the line problem with 1,024 agents when starting from scratch, a team trained with 256 agents could solve the problem at 1,024 when scaled up. Although it was not common, this result implies that scaling up may offer more than just the benefit of saving computational time, but also may allow multiagent HyperNEAT to solve problems that are prohibitively deceptive or complex. The impact of deceptive agent interactions for large teams may diminish at smaller team sizes where such interactions are not as devastating.

However, it is true that multiagent HyperNEAT could not always scale up perfectly post-training. As described above, there are some cases where the policies required to solve a larger size are fundamentally different than those at smaller sizes. This observation separates these results from previous multiagent HyperNEAT scaling attempts [25] in which the size of the problem (that is, the number of prey), and therefore the strategy required to solve the problem, remained constant even as team size increased. However, even when the number of prey increases as in this paper, the team at the smaller size may still encode fundamental regularities that are important to the problem regardless of scale, such as symmetry or the ability to flank a prey. That way, this approach to scaling can still be a good starting heuristic for additional training. Indeed, even when HyperNEAT's scaling was not perfect, some teams could nevertheless rapidly adapt to a new size when explicitly trained further for it. Thus, while the policy geometry discovered at smaller team sizes may overspecialize or may contain artifacts that were not sampled at the training size, the general patterns remain useful at different team sizes and can be quickly adapted for such different sizes. Sarsa(λ) was not able to benefit from this capability because even though it can continue training at new sizes as well, it is blind to the geometry of the team and therefore cannot intelligently extrapolate the behavior of the team as a whole.

In addition to scaling up, teams trained with multiagent HyperNEAT were able to scale down, which could allow recalibrating a team after some agents have been damaged. Scaling down also exemplifies the need for policy geometry because the regularities discovered at larger team sizes can be maintained at smaller sizes. However, scaling down did not always work because the algorithm may have converged onto regularities that are only useful at larger team sizes, or may have relied on a specific agent position that no longer exists. Nevertheless, the ability of a team to sometimes dynamically grown or shrink without further learning is a beneficial feature imparted by multiagent HyperNEAT.

The main result is that teams that were seeded and trained by multiagent HyperNEAT are the most effective. Such teams exploit the continuum of heterogeneity to overcome the problem of reinvention by starting with an existing useful policy and varying it only as much as is needed through a pattern across the team's geometry. Accordingly, the typical results (Sect. 5.2) show the dramatic role such regularities play in solutions at large sizes (which Sarsa(λ) could not solve). The major contributions of this paper are thus (1) to introduce the idea of *policy geometry* and show how it can be encoded and exploited to allow scaling, (2) to introduce the *continuum of heterogeneity* as a response to the problem of reinvention, and (3) to compare a method that takes advantage of policy geometry and the continuum of heterogeneity to one that does not. The hope is that these new concepts and approaches mark the beginning of a significant new direction in multiagent learning research that complements prior approaches.

Indeed, an unfortunate result of comparison-driven experiments is often an unwarranted emphasis on determining the *better* method. While multiagent HyperNEAT indeed exhibits greater scalability both during and after training, of course it was *designed* from the ground up (that is, through the scalability of indirect encoding in HyperNEAT) to be able to scale. On the other hand, Sarsa(λ) was not designed with this aim in mind, and thus does not acquire it in the multiagent case. At the same time, as an evolutionary approach, HyperNEAT does not offer the online learning capability inherent in value-function-based reinforcement learning such as Sarsa(λ). Thus rather than a critique of Sarsa(λ), this comparison is better interpreted as a validation that multiagent HyperNEAT brings something new to the table through the idea of policy geometry. In this more cooperative spirit, perhaps a more enlightened approach to interpreting comparisons is to consider that the fruits of divergent research areas may sometimes be complementary, and thus present opportunities for cross-fertilization. However, such approaches would likely be non-trivial, for example, beyond simply adding initial position to an agent's state space. The emphasis on scalability and large size that has been a focus of research in indirect encoding within evolutionary computation for many years has of yet attracted little attention in the reinforcement learning community in general. Perhaps this study can help to begin to bridge this gap by showing what we all might gain by beginning to communicate despite our foundational differences.

For example, an intriguing possibility is that indirect encodings such as CPPNs can be combined somehow with traditional approaches. That is, perhaps MARL can work at the level of a team instead of on individual agents and reinforcement signals can modify the weights of the CPPN. Yet this prospect can only be realized if a method is devised to propagate the reinforcement error signal through the level of indirection between the substrate ANN and the CPPN that encodes it. At present, no such *indirect error propagation* technique exists, though the possibility is open. Thus, at present, multiagent HyperNEAT is unique in its capability to scale teams through policy geometry.

In this sense, the major contribution of this work is conceptual because it offers a novel perspective on multiagent learning. In their recent survey of cooperative multiagent learning, Panait and Luke [67] cite scalability as a "major open topic" in the field and go on to say, "The 'multi' in multi-agent learning cries out for larger numbers of agents, in the range of ten to thousands or more. Two- and three-agent scenarios are reasonable simplifications to make theoretical analysis feasible: but the experimental and empirical literature ought to strive for more." This paper has shown that perhaps the impediment has been that it is simply impractical from the traditional perspective of multiagent learning to strive towards such large sizes given that each doubling of team size required Sarsa(λ) to perform an order of magnitude more evaluations to find a solution. However, by taking an alternative approach and exploiting the ideas of policy geometry and the continuum of heterogeneity, multiagent HyperNEAT was able to create cooperative heterogeneous teams of significant size that could scale without additional learning, which directly responds to this calling.

7 Conclusion

This paper presented a new method of training multiagent teams called multiagent HyperNEAT that overcomes the *problem of reinvention* faced by multiagent learning by exploiting team geometry and the continuum of heterogeneity. Multiagent HyperNEAT accomplishes these goals by representing teams as a pattern of policies, rather than as several distinct agents. Representing teams in this way also affords multiagent HyperNEAT the ability to scale team sizes dynamically up to several orders of magnitude larger than the size on which they were trained, a novel and powerful capability for heterogeneous teams. When compared against the traditional learning method multiagent Sarsa(λ), multiagent HyperNEAT significantly outperformed it in both training and scaling. Ultimately multiagent HyperNEAT offers a new perspective on multiagent learning that focuses on how agents on a team relate to one another and how those relationships can be exploited to foster cooperation.

Acknowledgments This material is based upon work supported by DARPA through grants HR0011-08-1-0020, HR0011-09-1-0045 and N11AP20003 (Computer Science Study Group Phases 1, 2, and 3), and the US Army Research Office under Award No. W911NF-11-1-0489. It does not necessarily reflect the position or policy of the government, and no official endorsement should be inferred.

References

1. Aaltonen et al (over 100 authors) (2009) Measurement of the top quark mass with dilepton events selected using neuroevolution at CDF. Phys Rev Lett 102(15):2001
2. Altenberg L (1994) Evolving better representations through selective genome growth. In: Proceedings of the IEEE world congress on computational intelligence. IEEE Press, Piscataway, NJ, pp 182–187
3. Angeline PJ, Saunders GM, Pollack JB (1993) An evolutionary algorithm that constructs recurrent neural networks. IEEE Trans Neural Netw 5:54–65
4. Baldassarre G, Trianni V, Bonani M, Mondada F, Dorigo M, Nolfi S (2007) Self-organized coordinated motion in groups of physically connected robots. IEEE Trans Syst Man Cybern Part B Cybern 37(1):224–239
5. Bentley PJ, Kumar S (1999) 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). Kaufmann, San Francisco, pp 35–43
6. Bongard J (2000) Reducing collective behavioural complexity through heterogeneity. Artificial life VII: proceedings of the seventh international conference on artificial life
7. Bongard JC (2002) Evolving modular genetic regulatory networks. In: Proceedings of the 2002 congress on evolutionary computation
8. Bousquet F, Le Page C (2004) Multi-agent simulations and ecosystem management: a review. Ecol Model 176(3–4): 313–332
9. Boutilier C (1996) Planning, learning and coordination in multiagent decision processes. In: Proceedings of the 6th conference on theoretical aspects of rationality and knowledge. Morgan Kaufmann Publishers Inc., pp 195–210
10. Bowling M, Veloso M (2002) Multiagent learning using a variable learning rate. Artif Intell 136(2):215–250
11. Bryant BD, Miikkulainen R (2003) Neuroevolution for adaptive teams. In: Proceedings of the 2003 congress on evolutionary computation (CEC 2003), vol 3. IEEE, Piscataway, NJ, pp 2194–2201
12. Bull L, Holland O (1997) Evolutionary computing in multiagent environments: eusociality. In: Proceedings of the annual conference on genetic programming. Morgan Kaufmann
13. Busoniu L, Schutter BD, Babuska R (2005) Learning and coordination in dynamic multiagent systems. Technical Report 05-019, Delft University of Technology
14. Busoniu L, Babuška R, De Schutter B (2008) A comprehensive survey of multi-agent reinforcement learning. IEEE Trans Syst Man Cybern Part C Appl Rev 38(2):156–172. doi:10.1109/TSMCC.2007.913919
15. Castelpietra C, Iocchi L, Nardi D, Piaggio M, Scalzo A, Sgorbissa A (2000) Coordination among heterogeneous robotic soccer players. In: Intelligent robots and systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ international conference on, IEEE, vol 2, pp 1385–1390
16. Christensen A, Dorigo M (2006) Incremental evolution of robot controllers for a highly integrated task. From animals to animats 9, pp 473–484
17. Claus C, Boutilier C (1998) The dynamics of reinforcement learning in cooperative multiagent systems. In: Proceedings of the national conference on artificial intelligence. John Wiley & Sons Ltd, pp 746–752
18. Clune J, Ofria C, Pennock R (2008) How a generative encoding fares as problem-regularity decreases. In: Proceedings of the 10th international conference on parallel problem solving from nature (PPSN 2008). Springer, Berlin, pp 258–367
19. Clune J, Beckmann BB, Pennock R, Ofria C (2009a) HybrID: a hybridization of indirect and direct encodings for evolutionary computation. In: Proceedings of the European conference on artificial life (ECAL-2009)
20. Clune J, Beckmann BE, Ofria C, Pennock RT (2009b) Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In: Proceedings of the IEEE congress on evolutionary computation (CEC-2009) special session on evolutionary robotics. IEEE Press, Piscataway, NJ, USA
21. Clune J, Pennock RT, Ofria C (2009) The sensitivity of HyperNEAT to different geometric representations of a problem. In: Proceedings of the genetic and evolutionary computation conference (GECCO-2009). ACM Press, New York, NY, USA
22. Clune J, Beckmann B, McKinley P, Ofria C (2010) Investigating whether HyperNEAT produces modular neural networks. In: Proceedings of the genetic and evolutionary computation conference (GECCO-2010). ACM Press, New York, NY
23. Conitzer V, Sandholm T (2007) AWESOME: a general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. Mach Learn 67(1): 23–43
24. Cybenko G (1989) Approximation by superpositions of a sigmoidal function. Math Control Signals Syst 2(4):303–314
25. D'Ambrosio D, Lehman J, Risi S, Stanley KO (2010) Evolving policy geometry for scalable multiagent learning. In: Proceedings of the ninth international conference on autonomous agents and multiagent systems (AAMAS-2010), international foundation for autonomous agents and multiagent system, pp 731–738
26. D'Ambrosio DB, Stanley KO (2008) Generative encoding for multiagent learning. In: Proceedings of the genetic and evolutionary computation conference (GECCO 2008). ACM Press, New York, NY
27. D'Ambrosio DB, Lehman J, Risi S, Stanley KO (2010) Evolving policy geometry for scalable multiagent learning. In: Proceedings of the 9th international conference on autonomous agents and multiagent systems: volume 1-volume 1, international foundation for autonomous agents and multiagent systems, pp 731–738
28. Drchal J, Koutnk J, Snorek M (2009) HyperNEAT controlled robots learn to drive on roads in simulated environment. In: Proceedings of the IEEE congress on evolutionary computation (CEC-2009). IEEE Press, Piscataway, NJ, USA
29. Dupuy TN (1990) The evolution of weapons and warfare. Da Capo, New York, NY, USA
30. Eggenberger P (1997) Evolving morphologies of simulated 3d organisms based on differential gene expression. Fourth European conference on artificial life
31. Ficici S, Pollack J (2000) A game-theoretic approach to the simple coevolutionary algorithm. Lecture notes in computer science, pp 467–476
32. Floreano D, Dürr P, Mattiussi C (2008) Neuroevolution: from architectures to learning. Evol Intell 1:47–62
33. Gauci J, Stanley KO (2007) Generating large-scale neural networks through discovering geometric regularities. In: Proceedings of the genetic and evolutionary computation conference (GECCO 2007). ACM Press, New York, NY
34. Gauci J, Stanley KO (2008) A case study on the critical role of geometric regularity in machine learning. In: Proceedings of the twenty-third AAAI conference on artificial intelligence (AAAI-2008). AAAI Press, Menlo Park, CA
35. Gauci J, Stanley KO (2010) Autonomous evolution of topographic regularities in artificial neural networks. Neural Comput 22(7):1860–1898
36. Gauci J, Stanley KO (2010) Indirect encoding of neural networks for scalable go. In: Schaefer R, Cotta C, Kołodziej J, Rudolph G (eds) Parallel problem solving from nature—PPSN XI, vol 6238. Springer, Lecture Notes in Computer Science, pp 354–363
37. Gomez F, Miikkulainen R (1999) Solving non-Markovian control tasks with neuroevolution. In: Proceedings of the 16th international joint conference on artificial intelligence. Kaufmann, San Francisco, pp 1356–1361
38. Green C (2003–2006) SharpNEAT homepage. http://sharpneat.sourceforge.net/
39. Gruau F, Whitley D, Pyeatt L (1996) A comparison between cellular encoding and direct encoding for genetic neural networks. In: Koza JR, Goldberg DE, Fogel DB, Riolo RL (eds) Genetic programming 1996: proceedings of the first annual conference. MIT Press, Cambridge, MA, pp 81–89
40. Haasdijk E, Rusu A, Eiben A (2010) HyperNEAT for locomotion control in modular robots. Evolvable systems: from biology to hardware, pp 169–180

41. Harvey I (1993) The artificial evolution of adaptive behavior. PhD thesis, School of Cognitive and Computing Sciences, University of Sussex, Sussex
42. Haynes T, Sen S (1996) Co-adaptation in a team. Int J Comput Intell Organ 1(4):1–20
43. Hornby GS, Pollack JB (2002) Creating high-level components with a generative representation for body-brain evolution. Artif Life 8(3)
44. Hotz P, Gomez G, Pfeifer R (2003) Evolving the morphology of a neural network for controlling a foveating retina-and its test on a real robot. In: Artificial life VIII-8th international conference on the simulation and synthesis of living systems, vol 2003
45. Hsu W, Gustafson S (2002) Genetic programming and multi-agent layered learning by reinforcements. In: Genetic and evolutionary computation conference, pp 764–771
46. Hu J, Wellman M (2003) Nash Q-learning for general-sum stochastic games. J Mach Learn Res 4:1039–1069
47. Hu J, Wellman MP (1998) Multiagent reinforcement learning: theoretical framework and an algorithm. In: Proceedings of 15th international conference on machine learning. Morgan Kaufmann, San Francisco, CA, pp 242–250
48. Iba H (1996) Emergent cooperation for multiple agents using genetic programming. Parallel problem solving from nature PPSN IV, pp 32–41
49. Ishiwaka Y, Sato T, Kakazu Y (2003) An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning. Robot Auton Syst 43(4):245–256
50. Jim K, Giles C (2000) Talking helps: evolving communicating agents for the predator-prey pursuit problem. Artif Life 6(3):237–254
51. Kalech M, Kaminka G (2003) On the design of social diagnosis algorithms for multi-agent teams. In: International joint conference on artificial intelligence, vol 18, pp 370–375
52. Knoester D, Goldsby H, McKinley P (2010) Neuroevolution of mobile ad hoc networks. In: Proceedings of the 12th annual conference on genetic and evolutionary computation. ACM, pp 603–610
53. Kobayashi K, Nakano K, Kuremoto T, Obayashi M (2010) A state predictor-based reinforcement learning system. Electron Commun Jpn 93(6):8–18
54. Kok J, Hoen P, Bakker B, Vlassis N (2005) Utile coordination: learning interdependencies among cooperative agents. In: Proceeding symposium on computational intelligence and games, pp 29–36
55. Koza JR, Rice JP (1991) Genetic generalization of both the weights and architecture for a neural network. In: Proceedings of the international joint conference on neural networks, vol 2 (New York, NY). IEEE, Piscataway, NJ, pp 397–404
56. Kutschinski E, Uthmann T, Polani D (2003) Learning competitive pricing strategies by multi-agent reinforcement learning. J Econ Dyn Control 27(11–12):2207–2218
57. Lindenmayer A (1974) Adding continuous components to L-systems. In: Rozenberg G, Salomaa A (eds) L systems, Lecture Notes in Computer Science 15. Springer, Heidelberg, Germany, pp 53–68
58. Littman ML (1994) Markov games as a framework for multi-agent reinforcement learning. In: Machine learning: proceedings of the 11th annual conference. Kaufmann, San Francisco, pp 157–163
59. Luke S, Spector L (1996) Evolving graphs and networks with edge encoding: preliminary report. In: Koza JR (ed) Late-breaking papers of genetic programming 1996, Stanford Bookstore
60. Martin AP (1999) Increasing genomic complexity by gene duplication and the origin of vertebrates. Am Nat 154(2):111–128
61. Matarić M (1997) Reinforcement learning in the multi-robot domain. Auton Robots 4(1):73–83
62. Miconi T (2003) When evolving populations is better than coevolving individuals: the blind mice problem. In: Gottlob G, Walsh T (eds) Proceedings of the eighteenth international joint conference on artificial intelligence (IJCAI '03). Morgan Kaufmann
63. Miller JF (2004) Evolving a self-repairing, self-regulating, French flag organism. In: Proceedings of the genetic and evolutionary computation conference (GECCO-2004). Springer, Berlin
64. Montana DJ, Davis L (1989) Training feedforward neural networks using genetic algorithms. In: Proceedings of the 11th international joint conference on artificial intelligence. Kaufmann, San Francisco, pp 762–767
65. Nolfi S, Floreano D (1998) Coevolving predator and prey robots: do arms races arise in artificial evolution? Artif Life 4(4):311–335
66. Oliveira E, Fischer K, Stepankova O (1999) Multi-agent systems: which research for which applications. Robotics Auton Syst 27(1):91–106
67. Panait L, Luke S (2005) Cooperative multi-agent learning: the state of the art. Auton Agents Multi Agent Syst 3(11):383–434. doi:10.1007/s10458-005-2631-2
68. Panait L, Wiegand R, Luke S (2003) Improving coevolutionary search for optimal multiagent behaviors. Proceedings of the eighteenth international joint conference on artificial intelligence (IJCAI), pp 653–658
69. Panait L, Luke S, Harrison JF (2006) Archive-based cooperative coevolutionary algorithms. In: Proceedings of the 8th annual conference on genetic and evolutionary computation. ACM, New York, NY, USA, pp 345–352
70. Panait L, Luke S, Wiegand R (2006) Biasing coevolutionary search for optimal multiagent behaviors. IEEE Trans Evol Comput 10(6):629–645
71. Panait L, Tuyls K, Luke S (2008) Theoretical advantages of lenient learners: an evolutionary game theoretic perspective. J Mach Learn Res 9:423–457
72. Potter M, De Jong K (1994) A cooperative coevolutionary approach to function optimization. Lect Notes Comput Sci 866: 249–259
73. Potter M, Meeden L, Schultz A (2001) Heterogeneity in the coevolved behaviors of mobile robots: the emergence of specialists. In: International joint conference on artificial intelligence, vol 17. Lawrence Erlbaum Associates Ltd, pp 1337–1343
74. Potter MA, De Jong KA, Grefenstette JJ (1995) A coevolutionary approach to learning sequential decision rules. In: Eshelman LJ (ed) Proceedings of the sixth international conference on genetic algorithms. Kaufmann, San Francisco
75. Price B, Boutilier C (1999) Implicit imitation in multiagent reinforcement learning. In: Machine learning. Morgam Kaufmann Publishers, Inc., pp 325–334
76. Puppala N, Sen S, Gordin M (1998) Shared memory based cooperative coevolution. In: Evolutionary computation proceedings, 1998. IEEE world congress on computational intelligence., The 1998 IEEE international conference on, pp 570–574
77. Quinn M, Smith L, Mayley G, Husbands P, Quinn M, Smith L, Mayley G, Husbands P (2003) Evolving controllers for a homogeneous system of physical robots: structured cooperation with minimal sensors. Philos Trans R Soc Lond A Math Phys Eng Sci 361(1811):2321–2343
78. Ren Z, Williams AB (2003) Lessons learned in single-agent and multiagent learning with robot foraging. In: IEEE international conference on systems, man and cybernetics, 2003, vol 3, pp 2757–2762
79. Risi S, Stanley KO (2010) Indirectly encoding neural plasticity as a pattern of local rules. In: Proceedings of the 11th international conference on simulation of adaptive behavior (SAB2010). Springer, Berlin
80. Saravanan N, Fogel DB (1995) Evolving neural control systems. IEEE expert, pp 23–27
81. Schlachter F, Schwarzer C, Kernbach S, Michiels N, Levi P (2010) Incremental online evolution and adaptation of neural

networks for robot control in dynamic environments. In: ADAPTIVE 2010, the second international conference on adaptive and self-adaptive systems and applications, pp 111–116
82. Secretan J, Beato N, D'Ambrosio DB, Rodriguez A, Campbell A, Stanley KO (2008) Picbreeder: evolving pictures collaboratively online. In: CHI '08: proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems. ACM, New York, NY, USA, pp 1759–1768, doi: 10.1145/1357054.1357328
83. Secretan J, Beato N, D'Ambrosio DB, Rodriguez A, Campbell A, Folsom-Kovarik JT, Stanley KO (2011) Picbreeder: a case study in collaborative evolutionary exploration of design space. Evol Comput 19(3):373–403
84. Servin A, Kudenko D (2008) Multi-agent reinforcement learning for intrusion detection. Lect Notes Comput Sci 4865:211
85. Shoham Y, Powers R, Grenager T (2004) Multi-agent reinforcement learning: a critical survey. In: AAAI fall symposium on artificial multi-agent learning
86. Sims K (1994) Evolving 3D morphology and behavior by competition. In: Brooks RA, Maes P (eds) Proceedings of the fourth international workshop on the synthesis and simulation of living systems (Artificial Life IV). MIT Press, Cambridge, MA, pp 28–39
87. Singh S, Kearns M, Mansour Y (2000) Nash convergence of gradient dynamics in general-sum games. In: In Proceedings of the sixteenth conference on uncertainty in artificial intelligence
88. Soltoggio A, Bullinaria AJ, Mattiussi C, Dürr P, Floreano D (2008) Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. In: Bullock S, Noble J, Watson R, Bedau M (eds) Proceedings of the eleventh international conference on artificial life (Alife XI). MIT Press, Cambridge, MA
89. Stanley KO (2007) Compositional pattern producing networks: a novel abstraction of development. Genet Program Evol Mach Special Issue Dev Syst 8(2):131–162
90. Stanley KO, Miikkulainen R (2002) Evolving neural networks through augmenting topologies. Evol Comput 10:99–127
91. Stanley KO, Miikkulainen R (2003) A taxonomy for artificial embryogeny. Artif Life 9(2):93–130
92. Stanley KO, Miikkulainen R (2004) Competitive coevolution through evolutionary complexification. J Artif Intell Res 21:63–100
93. Stanley KO, Bryant BD, Miikkulainen R (2005) Evolving neural network agents in the NERO video game. In: Proceedings of the IEEE 2005 symposium on computational intelligence and games
94. Stanley KO, Bryant BD, Miikkulainen R (2005) Real-time neuroevolution in the NERO video game. IEEE Trans Evol Comput Special Issue Evolut Comput Games 9(6):653–668
95. Stanley KO, Kohl N, Miikkulainen R (2005) Neuroevolution of an automobile crash warning system. In: Proceedings of the genetic and evolutionary computation conference
96. Stanley KO, D'Ambrosio DB, Gauci J (2009) A hypercube-based indirect encoding for evolving large-scale neural networks. Artif Life 15(2):185–212
97. Stone P, Sutton RS (2001) Scaling reinforcement learning toward RoboCup soccer. In: Proceedings of the 18th international conference on machine learning. Morgan Kaufmann, San Francisco, CA, pp 537–544
98. Stone P, Veloso M (2000) Layered learning. In: Machine learning: ECML 2000, pp 369–381
99. Stone P, Veloso M (2000) Multiagent systems: a survey from a machine learning perspective. Auton Robots 8(3):345–383
100. Stone P, Sutton RS, Singh SP (2001) Reinforcement learning for 3 vs. 2 keepaway. In: RoboCup 2000: Robot Soccer World Cup IV. Springer, London, UK, pp 249–258
101. Stone P, Sutton R, Kuhlmann G (2005) Reinforcement learning for robocup soccer keepaway. Adapt Behav 13(3):165
102. Suematsu N, Hayashi A (2002) A multiagent reinforcement learning algorithm using extended optimal response. In: Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1. ACM, New York, NY, USA, pp 370–377
103. Sutton R (1996) Generalization in reinforcement learning: successful examples using sparse coarse coding. In: Advances in neural information processing systems 8, pp 1038–1044
104. Sutton R, Barto A (1998) Reinforcement learning: an introduction. The MIT press, Cambride, MA
105. Sutton RS (2009) Tile coding software, version 2.0, http://webdocs.cs.ualberta.ca/~sutton/tiles2.html
106. Talvitie E, Singh S (2007) An experts algorithm for transfer learning. In: Proceedings of the twentieth international joint conference on artificial intelligence, pp 1065–1070
107. Tan M (1997) Multi-agent reinforcement learning: independent vs. cooperative agents. Readings in agents, pp 487–494
108. Taylor M, Stone P (2009) Transfer learning for reinforcement learning domains: a survey. J Mach Learn Res 10:1633–1685
109. Taylor M, Whiteson S, Stone P (2007) Transfer via inter-task mappings in policy search reinforcement learning. In: Proceedings of the 6th international joint conference on autonomous agents and multiagent systems, pp 1–8. ACM
110. Taylor ME, Stone P (2005) Behavior transfer for value-function-based reinforcement learning. In: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems. ACM, New York, NY, USA, AAMAS '05, pp 53–59, doi:10.1145/1082473.1082482
111. Taylor ME, Whiteson S, Stone P (2006) Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In: GECCO 2006: proceedings of the genetic and evolutionary computation conference, pp 1321–1328
112. Verbancsics P, Stanley KO (2010) Evolving static representations for task transfer. J Mach Learn Res (JMLR) 11:1737–1769
113. Verbancsics P, Stanley KO (2010) Task transfer through indirect encoding. In: Proceedings of the genetic and evolutionary computation conference (GECCO 2010). ACM Press, New York, NY
114. Waibel M, Keller L, Floreano D (2009) Genetic team composition and level of selection in the evolution of multi-agent systems. IEEE Trans Evol Comput 13(3):648–660. doi:10.1109/TEVC.2008.2011741
115. Waskow SJ, Bazzan ALC (2010) Improving space representation in multiagent learning via tile coding. In: Proceedings of the 20th Brazilian conference on advances in artificial intelligence. Springer, Berlin, Heidelberg, SBIA'10, pp 153–162
116. Watson JD, Hopkins NH, Roberts JW, Steitz JA, Weiner AM (1987) Molecular biology of the gene, 4 edn. The Benjamin Cummings Publishing Company, Inc., Menlo Park, CA
117. Whiteson S, Kohl N, Miikkulainen R, Stone P (2005) Evolving keepaway soccer players through task decomposition. Mach Learn 59:5–30
118. Wiegand RP (2004) An analysis of cooperative coevolutionary algorithms. PhD thesis, George Mason University, Fairfax, VA, USA, director-Kenneth A. Jong
119. Woolley BG, Stanley KO (2010) Evolving a single scalable controller for an octopus arm with a variable number of segments. In: Schaefer R, Cotta C, Kołodziej J, Rudolph G (eds) Parallel problem solving from nature—PPSN XI, vol 6239. Springer, Lecture Notes in Computer Science, pp 270–279
120. Yao X (1999) Evolving artificial neural networks. Proc IEEE 87(9):1423–1447
121. Yong C, Miikkulainen R (2010) Co-evolution of role-based cooperation in multi-agent systems. IEEE Trans Auton Ment Dev 1:170–186