

Evolution of Controllers for Mobile Ad Hoc Networks

Technical Report MSU-CSE-10-9

David B. Knoester, Heather J. Goldsby, and Philip K. McKinley
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan 48824
{dk, hjg, mckinley}@cse.msu.edu

Authors' note: A version of this document without the appendix is to be published in the Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), 2010, under the title, "Neuroevolution of Mobile Ad Hoc Networks."

Abstract

This paper describes a study of the evolution of distributed behavior, specifically the control of agents in a mobile *ad hoc* network, using neuroevolution. In neuroevolution, a population of artificial neural networks (ANNs) are subject to mutation and natural selection. For this study, we compare three different neuroevolutionary systems: a direct encoding, an indirect encoding, and an indirect encoding that supports heterogeneity. Multiple variations of each of these systems were tested on a problem where agents were able to coordinate their collective behavior. Specifically, movement of agents in a simulated physics environment affected which agents were able to communicate with each other. The results of experiments indicate that this is a challenging problem domain for neuroevolution, and although direct and indirect encodings tended to perform similarly in our tests, the strategies employed by indirect encodings tended to favor stable, cohesive groups, while the direct encoding versions appeared more stochastic in nature.

1 Introduction

There are many examples in nature of communities of organisms that exhibit complex collective behaviors. For example, social insects build complex nests, ants alert each other of danger, fish school to avoid predation, and groups of humans engage in collective decision making. Many of these and other examples include *cooperation*, where individuals work towards a common purpose, as well as *coordination*, where individual efforts are integrated in a harmonious manner. For example, while starlings may coordinate (via flocking) to avoid predation, humans cooperate to make decisions – But this process is not necessarily harmonious!

All of these systems rely upon communication among individuals. This communication may take different forms, e.g., vocalization [1], chemical secretion [2], and alarm-drumming [3]. Systems that include cooperation, coordination, and communication may be classified as instances of *distributed behavior*, where a coherent global behavior emerges out of local interactions between individuals within the environment (c.f. biological self-organization [4]). Improving our understanding of distributed behaviors has the potential to advance not only the biological sciences, but also to aid in the design of computational systems.

Distributed behaviors are of interest to many different branches of engineering. For example, in *distributed problem solving*, a subfield of distributed artificial intelligence, multiple software agents are used to solve problems [5]. Distributed behaviors are also a key component of *distributed control*, where the individual tasks for a control problem are decentralized [6]. While traditional engineering approaches have been used to solve problems in domains such as information processing and multi-vehicle control [6], practitioners of evolutionary computation have employed various techniques for controlling teams of agents,

including cooperative coevolution [7] and both homogeneous and heterogeneous team evolution [8]. Compared to human-engineered solutions, evolutionary computation enables researchers to specify the desired global behaviors of the system directly and without *a priori* knowledge of required local behaviors, and may be particularly useful in systems that interact with real-world devices (called *cyber-physical* systems [9]).

In this study, we describe preliminary results of using different neuroevolutionary systems to discover distributed behaviors for a network of mobile autonomous communicating agents, similar to a mobile *ad hoc* network (MANET). Specifically, we evaluated multiple variations of three different neuroevolutionary systems (NEAT [10], HyperNEAT [11], and Multi-agent HyperNEAT [12]) on a coverage problem, where the agents were required to distribute themselves on a grid while maintaining network connectivity with each other. Each of these systems was used to evolve one or more artificial neural networks (ANNs), which were used as controllers for both the movement and communication behavior of agents within the network. This problem thus includes not only a functional component (i.e., “spread out to cover a region”), but also a non-functional component related to the topology of the network formed by the agents. Agents were provided with simulated radios, and were able to broadcast to their neighbors within a limited range. Whether agents were able to communicate with each other was defined by whether a neighbor is transmitting and also by distance between individuals.

The contributions of this work are as follows: First, using a variety of neuroevolutionary systems and configurations, we evolved ANNs that were able to control networks of mobile agents on a problem requiring coordination. Second, we compared the performance of the different neuroevolutionary systems and configurations and found that, in most cases, direct and indirect encodings performed similarly. Finally, we found that the primary factor determining difficulty of this problem was the number of agents being controlled, indicating that large-scale multi-agent control remains a general challenge for neuroevolutionary systems, regardless of encoding.

2 Related Work

From large-scale permanent networks such as the Internet to smaller temporary networks such as mobile *ad hoc* wireless networks (MANETs), network creation and topology maintenance algorithms are a fundamental aspect of distributed computing systems. In fixed networks, topology maintenance is typically performed by routers, while in *overlay networks*, a logical communication network is created and maintained by application-level software. In MANETs, the multitude of routing protocols available, and their varying tradeoffs related to reliability and overhead, complicate the engineering of reliable and mobile distributed systems. Methods that are able to adapt to the environment in which they are deployed, especially in those systems that interact with physical systems, are needed [9]. In the remainder of this section, we review related work in distributed control, multi-agent systems, and evolutionary computation.

Distributed problem solving and control. Traditionally a subfield of distributed artificial intelligence, *distributed problem solving* is the use of multiple, semi-autonomous, and cooperating software agents to solve a problem [5]. Occasionally, a distinction is made between decomposition/distribution approaches, and those that enable autonomous interactions among agents, known as multi-agent systems (MAS) [13]. Similarly, *distributed control*, also known as *cooperative control*, is primarily concerned with problems in multi-vehicle control, for example, hazardous material handling and mobile sensor networks [6]. The task of controlling the formation of a group of agents has been examined from various different perspectives, including that of graph-stability, switching hierarchical control strategies, adaptive gradient climbing, and many others [14]. In this work, instead of engineering control algorithms for the movement of agents, we employ neuroevolution to discover distributed behaviors for a mobile network.

Multi-agent Systems. In the broader context of multi-agent systems (MAS), the excellent taxonomy provided by Panait and Luke [13] places our study in the *team learning* category, where a single evolving individual learns the behavior for the entire group. While both NEAT and HyperNEAT-based approaches produce homogeneous teams, we note that in Multi-agent HyperNEAT, a single individual may produce heterogeneous teams. Moreover, the agents in this study engage in *direct communication* through message-passing, and no mechanism for indirect communication (e.g., stigmergy) is provided. Finally, the scenario

that we have selected for study is related to *cooperative target observation*, where a group of agents is tasked with collective observation of a target. The specifics of this scenario will be explained in more detail in Section 3.

Evolving coordination. Within evolutionary computation, numerous techniques have been proposed to evolve cooperative teams that solve tasks. Waibel, Keller, and Floreano provide an overview of the work done in this area [8]. Their classification highlights whether the group is homogeneous [15, 16] or heterogeneous [7] and whether selection is performed at the level of the individual [16] or team [7, 15]. Additionally, they compare the performance of four types of teams: (1) homogeneous teams developed using individual selection; (2) homogeneous teams developed using group selection; (3) heterogeneous teams developed using individual selection, and (4) heterogeneous teams developed using group selection, on four cooperative tasks. Their results indicate that the genetic composition of groups and level of selection are key factors in evolving groups that perform cooperative tasks. Dorigo et al. [17] and Baldassarre, Parisi, and Nolfi [18] have also studied coordinated behavior of robotic swarms where individuals were able to physically attach to one another. In contrast to these studies, which focus on aggregation and coordinated motion of connected agents, our study focuses on controlling independent agents for a task that requires coordination among distant agents.

Neuroevolution. Neuroevolution is a form of evolutionary computation where artificial neural networks are produced through mutation and natural selection. Through real-time interaction with a player, NERO enabled the evolution of teams of agents, where each agent was controlled by an instance of an evolving neural network [19]. Additionally, Yong and Miikkulainen [7] have applied a cooperative coevolutionary architecture to evolve coordinated predator behavior in a prey-capture task. Neuroevolution has also produced adaptive teams, where agents with identical neural networks nonetheless exhibit different behaviors [20]. In contrast to these, our study applies multiple neuroevolutionary systems to the problem of controlling agents in a MANET, where agents must both coordinate their movements with each other, as well as maintain network connectivity.

A closely related study is that by Hauert, Zufferey, and Floreano [21], where a neural network controller for unmanned aerial vehicles (UAVs) was evolved via a genetic algorithm. Here, the UAVs were to establish and maintain a multi-hop communications link between a base station and a target without global or agent-relative positioning information. The key differences between this study and our own are that we include explicit sensory information related to the relative positioning of agents and have focused on a coverage-related problem. Finally, D’Ambrosio et al. [22] investigated the use of HyperNEAT and Multi-agent HyperNEAT on a coverage-based problem where agents were able to sense the boundaries of their environment, but were not able to sense other agents. As described in the next section, here we study agents that are able to sense their relative positions in an environment that does not place limitations on movement.

3 Target Problem and Methods

This section describes the specific problem we addressed and the three neuroevolutionary systems used.

3.1 Grid-Coverage Problem

Figure 1 is an illustration of a mobile network for oceanic monitoring. Here, agents in the network must not only coordinate their movement in order to remain connected, but they must also spread out to maximize the coverage area of their sensors. As the first step towards evolving solutions to such problems, we have defined the *grid-coverage problem*, where a mobile network of autonomous agents is to be maneuvered into a grid-like formation within a virtual physical environment.

Figure 2(a) depicts the environment and initial configuration for a network whose behavior will be evolved via neuroevolution. As shown here, a 6×6 grid of cells, each 4×4 m in size, is arrayed in a flat environment comprising 16 mobile agents. Each agent is modeled as rolling sphere. The physics of this environment are calculated by the Open Dynamics Engine (ODE, version 0.11.1, <http://www.ode.org>). Within this

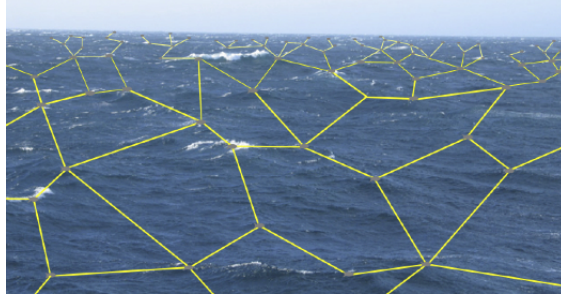
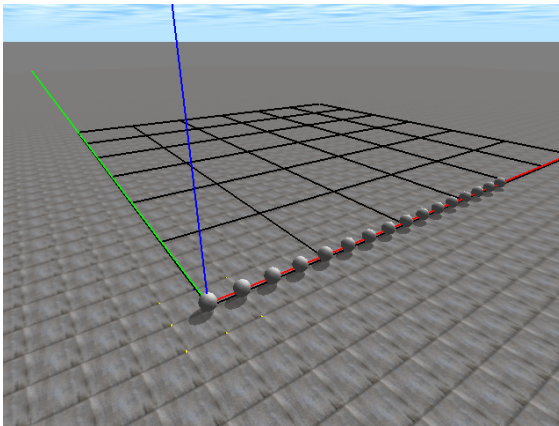
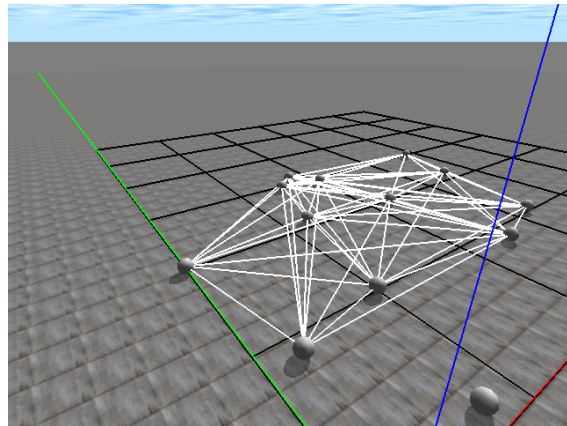


Figure 1: Illustration of a mobile sensor network for oceanic monitoring. Such a network could be used for monitoring oil spills and studies of ocean life.

environment, agents have a variety of sensors and effectors, summarized in Table 1. Figure 2(b) depicts this same network following 4s of control by an evolved neural network. Shown here are the communication links connecting agents in this network. Specifically, all agents that are within 10m of each other are able to communicate. When an agent transmits a message (by raising its tx effector above 0), a message is transmitted to neighbors within 10m of the sender. Receivers of the message have their appropriate receive sensor (rx) set to $1/distance$ of the sender to receiver. When multiple messages are received, the message from the nearest sender is used.



(a) Starting configuration of mobile sensors in for the grid-coverage problem.



(b) White lines represent connections between agents; agents can transmit data only to those other agents to which they are connected.

Figure 2: Starting configuration of agents (a) and position of agents 4s into a simulation (b). Grid agents are to cover is outlined with black lines; x , y , and z axis represented by red, green, and blue lines, respectively.

Within this environment, the fitness of a network of agents is calculated as:

$$F = \begin{cases} 1.0 & \text{if } |centroid| > 200 \\ 100 * Unique(A_v) & \text{if } |centroid| \leq 200 \end{cases}$$

where F is fitness, A_v is the subset of agents moving slower than a specified threshold (8m/s in this study) and $Unique$ counts the number of unique grid cells occupied by the agents. Fitness is calculated every 5s during a total simulation time of 30s, and the results are averaged. We note that if the network is unconnected at the time of fitness evaluation, a fitness of 1.0 (the minimum possible) is assigned for that evaluation. Furthermore, if the centroid of the network is ever located greater than 200m from the origin, a total fitness of 1.0 is immediately assigned and no further fitness evaluations are performed. These steps, as well as the pruning of fast-moving agents, were optimizations to reduce unnecessary computation time.

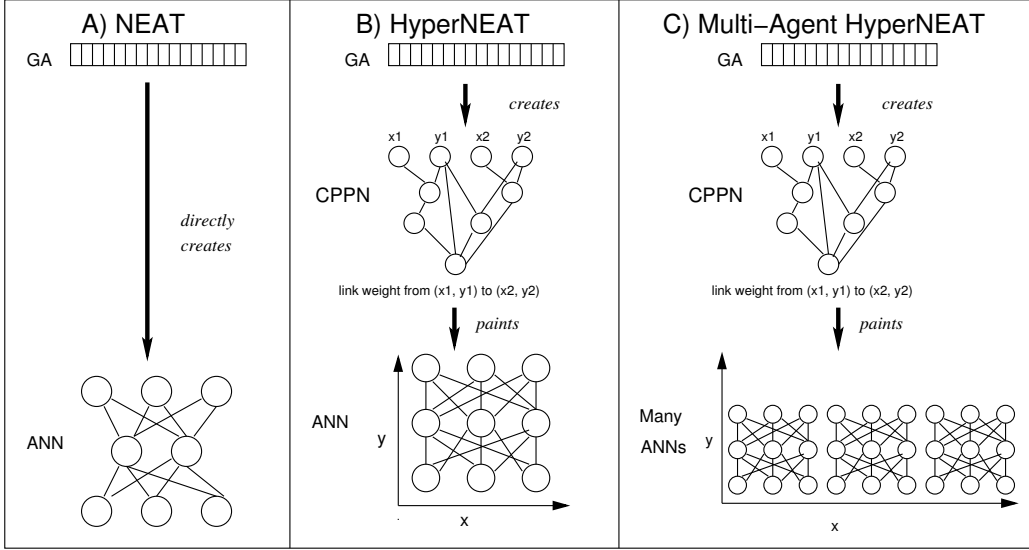


Figure 3: A graphical depiction of how NEAT, HyperNEAT, and Multi-agent HyperNEAT produce ANNs.

General solutions to network coverage problems require explicit coordination among agents. For example, a network of agents may become disconnected if they do not coordinate to avoid obstacles in their environment. In mobile sensor networks, where only a limited number of nodes have access to a base station, disconnectedness can have disastrous consequences on the effectiveness of the MANET. Our investigation of the grid-coverage problem is a first step in this direction. We note that in other systems, for example, predator-prey systems [12], the behavior of an individual may only loosely depend on the behavior of its neighbors. For example, agents may employ different strategies and even appear to cooperate, but without an explicit dependency on their neighbors.

We elected to study the grid-coverage problem due to its general structure. Although not presented here, the environment used for the grid-coverage problem can be easily adapted to search for other network characteristics. For example, we have already devised fitness functions that reward for the construction of networks with certain characteristic path lengths, diameters, clustering coefficients, and edge counts. Moreover, this problem can be made more difficult in a variety of ways, e.g., increasing the number of agents, removing agents during the simulation, altering the shape and location of the grid, introducing obstacles, or replacing the flat environment with wavy or mountainous terrain.

Table 1: Sensors and effectors of individual agents for the grid-coverage problem.

Name	Sensor/effector	Description
$rx_0 \dots rx_3$	sensor	directional radio strength; each sensor covers a 90° "pie slice" around the agent
$v_0 \dots v_2$	sensor	velocity vector (x, y, z)
$p_0 \dots p_2$	sensor	absolute position (x, y, z)
$f_0 \dots f_2$	effector	force vector (x, y, z)
tx	effector	transmit message

3.2 Neuroevolution Approaches

For this study into distributed control, we used three different neuroevolutionary systems, NEAT [10], HyperNEAT [11], and Multi-agent HyperNEAT [12], depicted in Figure 3. Each of these systems was used to evolve ANNs which were used as controllers for both the movement and communication behavior of agents within a mobile network.

NEAT. The first neuroevolutionary approach we used was NeuroEvolution of Augmenting Topologies (NEAT) [10]. NEAT is a genetic algorithm that uses a *direct encoding*, where each gene encodes a specific piece of the ANN – either a node or a connection between nodes (Figure 3a). One key advantage of NEAT is that it is able to evolve the structure of the ANN. Specifically, NEAT starts with ANNs that do not include a hidden layer. Over time, using a process called *complexification*, NEAT is able to evolve genomes that include nodes on hidden layers with varying connections. In this way, NEAT does not bias or constrain the solution to any particular topology.

HyperNEAT. Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) [11] is a genetic algorithm that uses an *indirect encoding*, where each gene encodes a specific piece of a Compositional Pattern Producing Network (CPPN); the CPPN, once fully constructed, produces the ANN (Figure 3b). Within HyperNEAT, a CPPN is effectively a complex mathematical function that takes as input the coordinates of a source node and target node and produces as output the weight for the link connecting them. By querying each possible source and target node, an ANN is produced from a CPPN. For evolutionary purposes, CPPNs are represented as graphs, where the nodes are mathematical functions such as gaussians, sigmoids, and sines, while the edges govern the composition relationship among the functions. The mathematical functions are capable of producing any ANN. HyperNEAT is an extension of NEAT that uses the NEAT algorithm to evolve CPPNs that produce ANNs, rather than directly evolving ANNs.

The primary advantage to HyperNEAT is that its use of an indirect encoding enables the creation of ANNs that exhibit symmetry and repeated motifs. Thus, HyperNEAT can capitalize on the regularity present in many domains and evolve solutions to more complex problems [23].

Multi-agent HyperNEAT. The third approach that we used was Multi-agent HyperNEAT [12], which is an extension of HyperNEAT that enables one genome to encode a set of ANNs that represent controllers for a heterogeneous team (Figure 3c). Specifically, Multi-agent HyperNEAT adds an additional input to the CPPN that represents which ANN is being evolved. The CPPN is then used to produce one ANN for each agent. As a result of the additional input, Multi-agent HyperNEAT is able to evolve ANNs that make use of many common strategy elements, but may also differ from one another [12].

Substrate Structure. A key part of using HyperNEAT-based systems is to determine the *substrate structure*, specifically the values to pass to the CPPN for each neuron. In practice, this means that each neuron must be placed in a Cartesian coordinate frame, and the coordinates of that neuron are then used as either the source or destination depending on the link being evaluated. For example, if we assume that the `rx0` radio sensor neuron is located at $(0, 0)$, and that it is linked to the `f0` force output neuron at $(2, 2)$, then the weight of link connecting these two neurons is: $w = CPPN(0, 0, 2, 2)$. As will be shown in Section 4, we tested multiple substrate structures.

4 Results

For all of the experiments described in this section, we used the fitness function described in Section 3.1, each agent had the sensors and effectors summarized in Table 1, and 30 trials were conducted for all treatments. Each treatment was executed for 200 generations on a population of size 100. Mutation rates for adding links and nodes to NEAT were 0.05 and 0.03, respectively. Mutation rates for adding links and nodes to the CPPN for HyperNEAT-based treatments were set equivalently. We note that this may allow differences in *effective* mutation rates between NEAT and HyperNEAT-based treatments.

4.1 Evolving Mobile Sensor Controllers

Our first experiment was designed to evaluate the performance of NEAT, HyperNEAT, and Multi-agent HyperNEAT on the grid-coverage problem. Initially, we used a 16-node swarm and naïve substrate structures for the HyperNEAT and Multi-agent HyperNEAT treatments. Specifically, we used a fixed topology that contained one hidden layer, where each node in the input layer was linked to each node in the hidden layer, which was in turn linked to each node in the output layer. NEAT, of course, evolves not only the topology of the ANN, but also connection weights, so no additional structures were required.

Figure 4 plots the fraction of maximum possible fitness achieved by NEAT (N), HyperNEAT (H1), and Multi-agent HyperNEAT (M1), averaged across all 30 trials. All three techniques achieved between 60% to 70% of maximum fitness before reaching a plateau. While treatment N had the overall best performance, tests for statistical significance revealed that treatment N differed from treatment H1, but no other pairs were significantly different from each other (*Kruskal-Wallis* multiple comparison, $p < 0.01$).

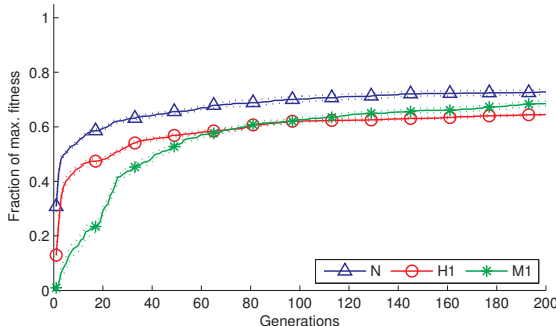


Figure 4: Fraction of maximum fitness achieved with NEAT (N), HyperNEAT (H1), and Multi-agent HyperNEAT (M1) on networks comprising 16 agents.

Previous research has demonstrated that the structure of the underlying substrate used by HyperNEAT affects its ability to effectively solve problems [24]. Selecting an inferior substrate can result in substandard performance. To ensure that the performance of HyperNEAT-based systems were not unfairly affected by the selection of a poor substrate structure, we next tested several additional configurations, summarized in Table 2. Details, including the specific assignment of neurons to coordinates, can be found in an accompanying technical report [25].

Figure 5 plots the fraction of maximum fitness of all NEAT- and HyperNEAT-based treatments. Each box represents a single treatment, and was constructed from the maximum fitness achieved by the best performing individual from each of the 30 trials. Treatments are positioned in ascending mean rank order from left to right. In general, the mean performance ranges from 60% to 70% of maximum fitness. Overall, while the maximum performance of NEAT exceeded that of all HyperNEAT-based treatments, no statistically significant differences between the highest performing treatments were found ($p < 0.01$; the “n.s.” annotation in Figure 5 indicates these treatments). Of the HyperNEAT-based treatments, the only one whose mean performance (though not mean rank) exceeded that of NEAT (treatment N) was M3, which employed Multi-agent HyperNEAT. Many of these treatments were also tested with an additional (second) hidden layer; results were not statistically significantly different from a single hidden layer (data not shown).

4.2 Key Contributors to Problem Difficulty

Our next set of experiments investigated some of the factors that contributed to the difficulty of the grid-coverage problem. In addition to the NEAT-based treatment (N), we selected two variants of HyperNEAT (D2) and Multi-agent HyperNEAT (M1) for further analysis. Here, we present the results of two of these investigations, specifically regarding the availability of sensor data and the number of agents included in the simulation.

Table 2: Substrate structures for HyperNEAT and Multi-agent HyperNEAT treatments used in this paper. Treatments H1-H4, D1-D5, F1, and T used HyperNEAT, while treatments R1-R2, M1-M3, and F2 used Multi-agent HyperNEAT.

Treatment	Description
H1-H4	Each layer of the neural network is given its own 2-D coordinate frame. Inputs to the CPPN for each neuron are a triple (x, y, l) , where x, y are the coordinates of the neuron, and l is the neural network layer. Each treatment uses a different substrate structure.
D1-D5	Each layer of the neural network is given its own 3-D coordinate frame. Inputs to the CPPN are a tuple (x, y, z, l) , where x, y, z are the coordinates of the neuron, and l is the neural network layer. Each treatment uses a different substrate structure.
M1-M3	Each neuron is described with a tuple (x, y, l, i) , where x, y are the coordinates of that neuron, l is the neural network layer, and i is a unique identifier based on the number of agents (e.g., for a 16 agent network, i ranges from $0 \dots 15$). M1 uses the substrate structure from H1, M2 uses the substrate structure from D1, and M3 adds a bias neuron to M1.
R1-R2	Each neuron is described with a tuple (x_g, y_g, x_r, y_r, l) , where x_g, y_g are the global coordinates for that neuron, and x_r, y_r are relative coordinates for that neuron, and l is the neural network layer. Relative coordinates were calculated by scaling x_g, y_g by i/m , where i is a unique identifier for that agent, and m is the total number of agents. R1 uses the substrate structure from H1, while R2 uses the substrate structure from D2.
F1-F2	Each neuron is described with a pair (x, y) , and all layers are placed on the same coordinate frame. Each treatment uses a different substrate structure.
T	CPPN is modified to return two weights, one for each link between layers; uses the same substrate structure as H1.

The first factor that impacted performance was the availability of sensor data. For these treatments, we zeroed out the relevant sensors and re-ran each treatment to produce new ANNs. Specifically, we ran one set of experiments where we removed the radio strength sensing capability and one where we removed the location sensing capability. We note that these results do not reveal whether the previously evolved ANNs used this information, but rather can be used to understand what sensor information is required to find general solutions to the grid-coverage problem.

Figures 6 and 7 depict the results of removing the radio sensing and location sensing capabilities, respectively, from all three approaches. The performance of HyperNEAT and Multi-agent HyperNEAT dropped significantly (up to 20%) compared to their performance in Figure 5, indicating that these sensors were most likely a key part of their previous solutions. The performance of NEAT was less affected by the loss of these sensors, indicating that the strategies evolved by NEAT were less dependent on their environment. In both cases, the performance of NEAT is significantly different from the HyperNEAT-based treatments ($p < 0.01$),

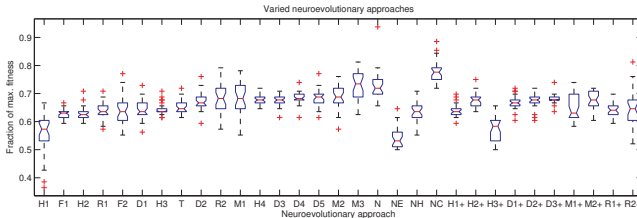


Figure 5: Fraction of maximum fitness achieved with NEAT, HyperNEAT, and Multi-agent HyperNEAT on networks comprising 16 agents. Multiple different substrate configurations were tested for HyperNEAT and Multi-agent HyperNEAT treatments. Approaches are in ascending mean rank order from left to right. Treatment N (far right) is the NEAT-based treatment; see Table 2 for descriptions of HyperNEAT-based treatments.

while the two HyperNEAT-based treatments were not significantly different from each other at the $p < 0.01$ level.

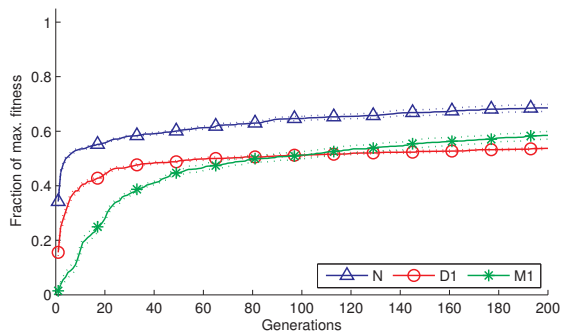


Figure 6: Performance of NEAT (N), HyperNEAT (D1), and Multi-agent HyperNEAT (M1) decrease when radio sensors are removed.

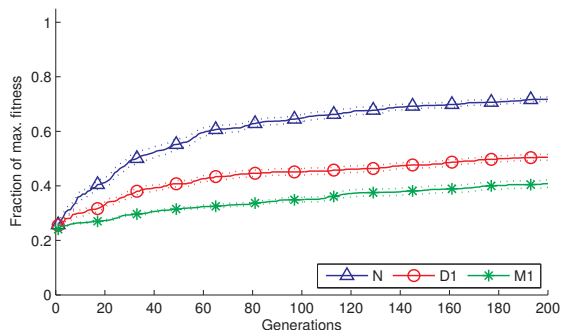


Figure 7: Performance of NEAT (N), HyperNEAT (D1), and Multi-agent HyperNEAT (M1) decrease when location sensors are removed.

While it is well-known that the number of agents being controlled is a significant factor for multi-agent systems [13], it has been conjectured that generative and indirect encodings might be better equipped to deal with this challenge than direct encodings [12]. To better understand how the number of agents might affect performance on the grid-coverage problem, we performed a series of additional treatments that varied the number of agents from 2 to 32. Figures 8, 9, and 10 depict the performance of NEAT, HyperNEAT, and Multi-agent HyperNEAT, respectively, as the number of agents is varied. The same pattern of performance is repeated across all three treatments, where we see that as the number of agents increases, overall performance steadily decreases. We note that in these figures, the performance of a given treatment is relative to the number of agents. Thus, while performance (measured as the fraction of maximum fitness) decreases as the number of agents is increased, the total area covered by the agents increases. Interestingly, the performance of the NEAT-based treatment degraded somewhat more slowly than the HyperNEAT-based treatments (mean performance drop between network sizes 11.5% for NEAT, 13.3% for HyperNEAT, $p < 0.01$).

4.3 Biological self-organization

A key component of biological self-organization is whether agents react to each others' behavior. In most of the experiments conducted, agents had the capability to modulate their communication behavior. Specifically, when the tx output was greater than 0.0, an agent broadcast that signal to its neighbors. A measure of the strength of that signal ($1/\text{distance}$) was then applied to the appropriate rx input. In our experiments,

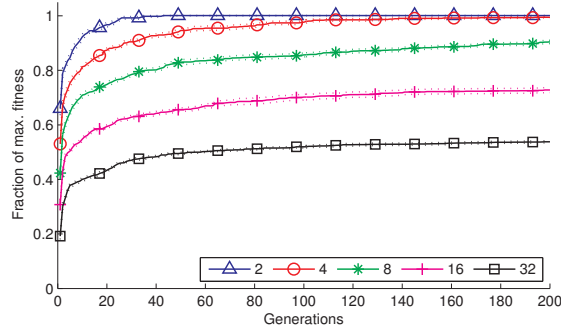


Figure 8: Performance of NEAT for different numbers of agents.

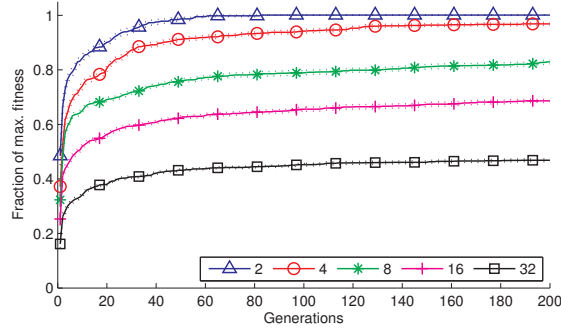


Figure 9: Performance of HyperNEAT for different numbers of agents.

we found a significant fitness difference between treatments that included this capability versus those that did not. However, this does not address the question of whether the treatments that included signal strength were actually using it. To test whether or not the controllers produced via neuroevolution were in fact engaging in self-organizing behaviors, we took the final solutions from each of NEAT, HyperNEAT, and Multi-agent HyperNEAT treatments and tested them in an environment where all rx inputs were set to 0.

Figure 11 plots the original fitness of each dominant vs. its fitness with their rx inputs set to 0. Points that are above the diagonal (all but 2) represent solutions whose fitness is greater when sensors are enabled. For example, the point near (0.5, 0.7) represents a single trial whose fitness with sensors enabled (the conditions under which it evolved) reached 70% of maximum fitness, while when its sensors are disabled (set to 0), it achieved only 50% of its maximum fitness. That all but two trials (both NEAT-based) had greater fitness

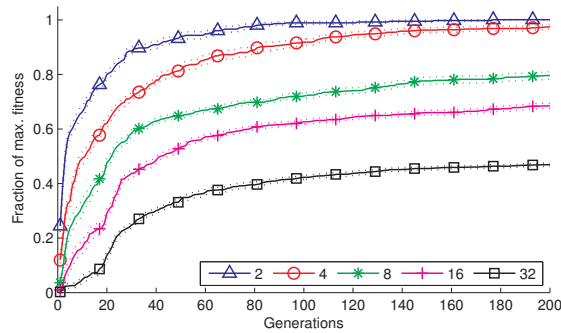


Figure 10: Performance of Multi-agent HyperNEAT for different numbers of agents.

with sensors than without is evidence of self-organization.

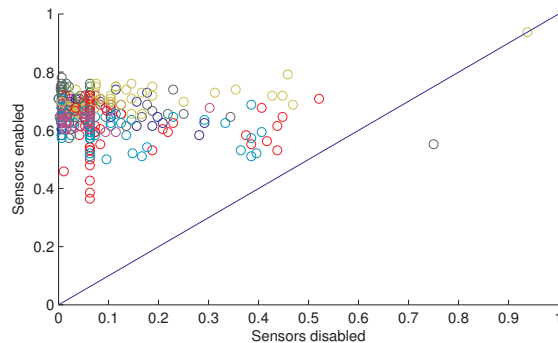


Figure 11: Fraction of maximum fitness with sensors enabled vs. sensors disabled. Each point in this plot represents a single trial; all treatments from Figure 5 are represented. Fitness is depressed when sensors are disabled, indicating that evolved solutions are self-organizing.

5 Conclusion

The experiments described in this paper demonstrate that neuroevolution can discover distributed behaviors for mobile sensor networks. On average, the highest-performing NEAT and HyperNEAT-based systems were able to achieve approximately 70% of the maximum possible grid coverage. In a network of 16 agents, this translates to approximately 5 agents inhabiting the same grid-cell as another, including during the first fitness evaluation where any movement patterns were likely still being established. Based on the results of different experimental treatments, we can conclude that the difficulty of coordinating this behavior is primarily a function of the number of agents in the network.

Qualitatively, we observed that agent behavior was a function of the neuroevolutionary system employed. Direct encodings seemed to prefer stochastic and/or non-self-organizing strategies, while the indirect encodings preferred self-organizing strategies. As in a related study [22], we also observed that Multi-agent HyperNEAT was able to discover suites of related strategies, where agents constructed by the same CPPN would behave somewhat differently.

The results presented here raise a number of interesting questions, related not only to the nature of the grid-coverage problem, but also to the relationship between NEAT and HyperNEAT-based systems. Our future work includes investigating why these systems perform similarly in this domain, and particularly if requiring explicit coordination among agents is a factor. We also plan to explore the evolution of distributed behavior in more complex environments. For example, it is yet unknown what effect communication failures may have on evolved strategies. Moreover, the tests presented in this paper were conducted in a “flat-world” environment. It is not clear what effect, if any, a more complex, e.g., wave-like or mountainous environment will have on the behavior of these various systems.

Acknowledgments. This work was supported in part by National Science Foundation grants CNS-0915885, CCF-0820220, and CNS-0751155; by U.S. Army Grant W911NF-08-1-0495; and by a Quality Fund Grant from Michigan State University. The authors would also like to acknowledge Jeff Clune, Benjamin E. Beckmann, and the anonymous reviewers for their insightful comments.

References

- [1] D. H. Owings and E. S. Morton, *Animal Vocal Communication: A New Approach*. Cambridge University Press, 1998.

- [2] C. M. Waters and B. L. Bassler, “Quorum sensing: Cell-to-cell communication in bacteria,” *Annual Review of Cell and Developmental Biology*, vol. 21, no. 1, 2005.
- [3] B. Holldobler and E. O. Wilson, *The Ants*. Harvard University Press, 1990.
- [4] S. Camazine, J. Deneubourg, N. Franks, J. Sneyd, G. Theraula, and E. Bonabeau, *Self-organization in biological systems*. Princeton University Press, 2003.
- [5] G. Weiss, ed., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.
- [6] W. Ren and R. W. Beard, *Distributed Consensus in Multi-vehicle Cooperative Control: Theory and Applications*. Springer, 2007.
- [7] C. Yong and R. Miikkulainen, “Cooperative coevolution of multi-agent systems,” Tech. Rep. AI-01-287, University of Texas at Austin, Austin, TX, February 2001.
- [8] M. Waibel, L. Keller, and D. Floreano, “Genetic Team Composition and Level of Selection in the Evolution of Cooperation,” *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 3, 2009.
- [9] W. Wolf, “Cyber-physical systems,” *IEEE Computer*, vol. 42, no. 3, 2009.
- [10] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” *Evolutionary Computation*, vol. 10, no. 2, 2002.
- [11] K. O. Stanley, D. B. D’Ambrosio, and J. Gauci, “A hypercube-based indirect encoding for evolving large-scale neural networks,” *Artificial Life*, vol. 15, no. 2, 2009.
- [12] D. B. D’Ambrosio and K. O. Stanley, “Generative encoding for multiagent learning,” in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2008.
- [13] L. Panait and S. Luke, “Cooperative multi-agent learning: The state of the art,” *Autonomous Agents and Multi-Agent Systems*, vol. 11, no. 3, 2005.
- [14] C. Cassandras and W. Li, “Sensor networks and cooperative control,” *European Journal of Control*, vol. 11, no. 4-5, 2005.
- [15] J. C. Bongard, “The Legion System: A novel approach to evolving heterogeneity for collective problem solving,” in *Proceedings of the European Conference on Genetic Programming*, pp. 16–28, 2000.
- [16] D. Floreano, S. Mitri, S. Magnenat, and L. Keller, “Evolutionary conditions for the emergence of communication in robots,” *Current Biology*, vol. 17, no. 6, 2007.
- [17] M. Dorigo, V. Trianni, E. Şahin, R. Groß, T. H. Labella, G. Baldassarre, S. Nolfi, J.-L. Deneubourg, F. Mondada, D. Floreano, and L. M. Gambardella, “Evolving self-organizing behaviors for a swarm-bot,” *Autonomous Robots*, vol. 17, no. 2, 2004.
- [18] G. Baldassarre, D. Parisi, and S. Nolfi, “Coordination and behavior integration in cooperating simulated robots,” in *Proceedings of the Conference on Simulation of Adaptive Behavior*, 2004.
- [19] K. Stanley, B. Bryant, and R. Miikkulainen, “Real-time neuroevolution in the NERO video game,” *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 6, 2005.
- [20] B. D. Bryant and R. Miikkulainen, “Neuroevolution for adaptive teams,” in *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 3, 2003.
- [21] S. Hauert, J.-C. Zufferey, and D. Floreano, “Evolved swarming without positioning information: an application in aerial communication relay,” *Autonomous Robots*, vol. 26, no. 1, 2009.

- [22] D. B. D'Ambrosio, J. Lehman, S. Risi, and K. O. Stanley, "Evolving policy geometry for scalable multiagent learning," in *Proceedings of the Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010.
- [23] J. Clune, C. Ofria, and R. Pennock, "How a generative encoding fares as problem-regularity decreases," in *Proceedings of the Conference on Parallel Problem Solving From Nature (PPSN)*, 2008.
- [24] J. Clune, C. Ofria, and R. Pennock, "The sensitivity of HyperNEAT to different geometric representations of a problem," in *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO)*, 2009.
- [25] D. B. Knoester, H. J. Goldsby, and P. K. McKinley, "Evolution of controllers for mobile ad hoc networks," Tech. Rep. MSU-CSE-10-9, Computer Science and Engineering, Michigan State University, East Lansing, Michigan, April 2010.
- [26] R. Brooks, "Artificial life and real robots," in *Proceedings of the European Conference on Artificial Life (ECAL)*, pp. 3–10, 1992.

A Experiment Details

This section describes the details of all experiments that we conducted as part of our study of the neuroevolution of controllers for mobile ad hoc networks. Table 1 summarizes the configuration differences among these experiments, while Figure 1 depicts their performance, measured as the average fraction of maximum fitness achieved over 30 trials. Treatment-specific information and details concerning the assignment of neurons to their locations in HyperNEAT substrates can be found in the following subsections.

Table 2 summarizes the common NEAT configuration parameters (these same parameters are used for the CPPN in HyperNEAT-based treatments as well). Except where noted, all treatments shared the same agent configuration. Physics simulation was conducted via the Open Dynamics Engine (ODE, version 0.11.1, <http://www.ode.org>). Version 2.5 of HyperNEAT (<http://eplex.cs.ucf.edu/software.html>) was used in all experiments (HyperNEAT includes the NEAT algorithm).

Table 1: Summary information for NEAT, HyperNEAT, and Multi-agent HyperNEAT experiments. Treatments H1-H4, D1-D5, F1, NE, and T used HyperNEAT; treatments R1-R2, M1-M3, and F2 used Multi-agent HyperNEAT; treatments N and NC used NEAT:

Treatment	Description
H1-H4	Each layer of the neural network is given its own 2-D coordinate frame. Inputs to the CPPN for each neuron are a triple (x, y, l) , where x, y are the coordinates of the neuron, and l is the neural network layer. Each treatment uses a different substrate structure.
D1-D5	Each layer of the neural network is given its own 3-D coordinate frame. Inputs to the CPPN are a tuple (x, y, z, l) , where x, y, z are the coordinates of the neuron, and l is the neural network layer. Each treatment uses a different substrate structure.
M1-M3	Each neuron is described with a tuple (x, y, l, i) , where x, y are the coordinates of that neuron, l is the neural network layer, and i is a unique identifier based on the number of agents (e.g., for a 16 agent network, i ranges from 0...15). M1 uses the substrate structure from H1, M2 uses the substrate structure from D1, and M3 adds a bias neuron to M1.
N	NEAT-based treatment; no substrate geometry needed.
NC	NEAT-based treatment that uses cell sensors.
NE	Substrates are configured as H3, however rx sensors read the locations of the nearest three neighbors instead of message strengths.
NH	As D3, without a hidden layer.
R1-R2	Each neuron is described with a tuple (x_g, y_g, x_r, y_r, l) , where x_g, y_g are the global coordinates for that neuron, and x_r, y_r are relative coordinates for that neuron, and l is the neural network layer. Relative coordinates were calculated by scaling x_g, y_g by i/m , where i is a unique identifier for that agent, and m is the total number of agents. R1 uses the substrate structure from H1, while R2 uses the substrate structure from D2.
F1-F2	Each neuron is described with a pair (x, y) , and all layers are placed on the same coordinate frame. Each treatment uses a different substrate structure.
T	CPPN is modified to return two weights, one for each link between layers; uses the same substrate structure as H1.

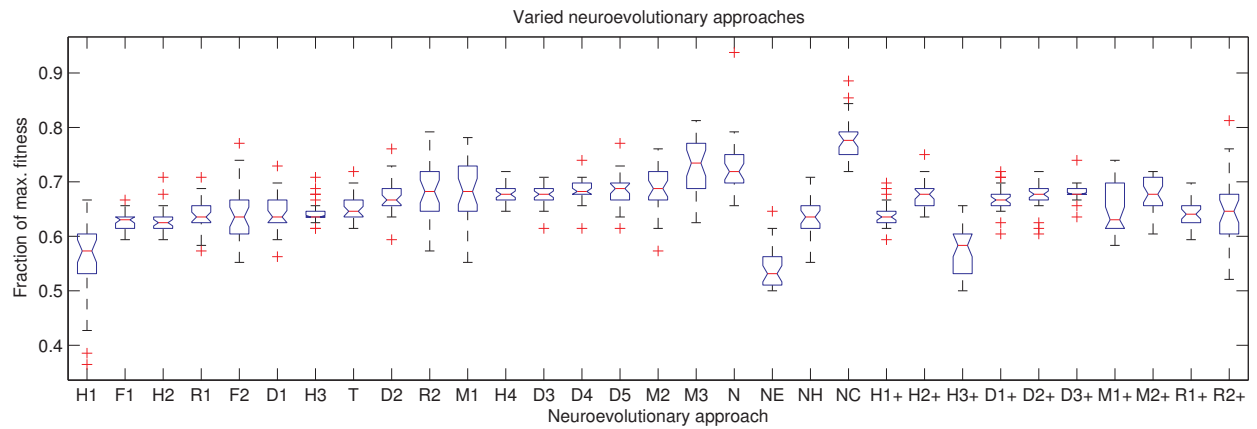


Figure 1: Performance, measured as the average fraction of maximum fitness achieved over 30 trials, for all NEAT, HyperNEAT, and Multi-agent HyperNEAT experiments. A “+” indicates an additional hidden layer.

Table 2: Configuration parameters common across all NEAT- and HyperNEAT-based experiments.

ConvergenceStep	1000.0
PopulationSize	100.0
MaxGenerations	200.0
DisjointCoefficient	2.0
ExcessCoefficient	2.0
WeightDifferenceCoefficient	1.0
FitnessCoefficient	0.0
CompatibilityThreshold	6.0
CompatibilityModifier	0.3
SpeciesSizeTarget	5.0
DropoffAge	15.0
AgeSignificance	1.0
SurvivalThreshold	0.2
MutateAddNodeProbability	0.03
MutateAddLinkProbability	0.05
MutateDemolishLinkProbability	0.03
MutateLinkWeightsProbability	0.8
MutateOnlyProbability	0.25
MutateLinkProbability	0.1
AllowAddNodeToRecurrentConnection	0.0
SmallestSpeciesSizeWithElitism	5.0
MutateSpeciesChampionProbability	0.0
MutationPower	1.5
AdultLinkAge	18.0
AllowRecurrentConnections	0.0
AllowSelfRecurrentConnections	0.0
ForceCopyGenerationChampion	1.0
LinkGeneMinimumWeightForPhentoype	0.0
GenerationDumpModulo	10.0
RandomSeed	7.0 (overridden on command-line)
ExtraActivationFunctions	1.0
AddBiasToHiddenNodes	0.0
SignedActivation	1.0
ExtraActivationUpdates	9.0
OnlyGaussianHiddenNodes	0.0
OnlySaveFinalPop	1
NumJointsWithError	0
WidthOfErrorRange	5.0
OnlySigmoidHiddenNodes	0.0

A.1 NEAT-based Experiments

Here we describe two different configurations of NEAT that we used on the grid-coverage problem.

Treatment N: Aside from defining the input, output, and bias neurons, no special considerations were needed to use NEAT on the grid-coverage problem. Although perhaps unsuitable for large neural networks where a generative encoding would be beneficial, the ease-of-use of NEAT is a great advantage.

Treatment NC: In this treatment we replace the directional rx sensors with cell sensors, where agents were able to directly sense the occupancy of their neighboring 8 cells. Although this improved performance, such sensors are unrealistic, and should only be used for comparison purposes [26].

A.2 HyperNEAT-based Experiments

Here we describe the different HyperNEAT-based configurations that we tested on the grid-coverage problem. Given that the substrate structure can have a significant impact on results [24], the configurations described here were designed to explore different substrate structures.

Treatment H1: This substrate structure was the first we tried, and was designed to be as simple as possible. Input neurons were arranged in columns in a 2-D coordinate frame for both the input and hidden layers, output neurons were arranged in a single column in the output layer. Position and velocity sensors were placed in the same row, and the x , y , and z elements of each were placed the same row. This treatment served as a baseline from which to construct other substrate structures. Indeed, subsequent substrate structures were constructed by altering characteristics of this first substrate, as will be seen.

Neuron	Location in substrate
v0...v2	(0, 0..2)
p0...p2	(0, 3..5)
rx0...rx3	(1, 0..3)
f0...f2	(0, 0..2)
tx	(0, 3)

Treatment H2: In this substrate, we split the position and velocity sensors into their own columns, under the expectation that the HyperNEAT CPPN will be able to better target these two types of sensors. Again, input neurons were arranged in columns in a 2-D coordinate frame for both the input and hidden layers, and output neurons were arranged in a single row in the output layer. Position and velocity sensors were placed in different columns.

Neuron	Location in substrate
v0...v2	(0, 0..2)
p0...p2	(1, 0..2)
rx0...rx3	(2, 0..3)
f0...f2	(0, 0..2)
tx	(0, 3)

Treatment H3: Here, instead of placing all *types* of sensors near each other, we placed all sensor *axes* near each other. For example, the x -axis sensor for velocity was placed near the x -axis position sensor, instead of near other velocity sensors. Input neurons were arranged in rows in a 2-D coordinate frame for both the input and hidden layers. Output neurons were placed in a single row.

Neuron	Location in substrate
v0...v2	((0, 4, 8), 1)
p0...p2	((1, 5, 9), 1)
rx0...rx3	((6, 2, 7, 3), 1)
f0...f2	((0, 4, 8), 2)
tx	(4, 0)

Treatment H4: This treatment modified H1 by placing sensors by row as opposed to by column. Specifically, all sensor types were placed in the same row, with the fourth rx sensor and tx output placed in their own rows.

Neuron	Location in substrate
v0...v2	(0..2, 0)
p0...p2	(0..2, 1)
rx0...rx3	(0..2, 2), (0, 3)
f0...f2	(0..2, 5)
tx	(-1, 6)

Treatment H5: In this treatment, different types of sensors were placed in different quadrants of the coordinate plane, while attempting to maintain a row-centric organization. It has been conjectured that HyperNEAT is better able to differentiate neurons that appear in different quadrants versus different coordinates in the same quadrant, so an arrangement such as this should improve performance relative to single-quadrant treatments.

Neuron	Location in substrate
v0...v2	(1..3, 1)
p0...p2	(1..3, -1)
rx0...rx3	(-1, 0), (-1, 1), (-2, 1), (-3, 1)
f0...f2	(0, 0..2)
tx	(0, 3)

Treatment D1: This treatment was the first that employed a three-dimensional layer structure (that is, each layer of the substrate was itself a 3D coordinate plane - all previous treatment were 2D). Different sensor types were placed by column, except for the rx sensors and f outputs, which were placed such that they surrounded the point (1, 1, z).

Neuron	Location in substrate
v0...v2	(0, 0..2, 0)
p0...p2	(0, 0..2, 1)
rx0...rx3	(2, 1, 2), (2, 2, 2), (0, 2, 2), (0, 1, 2)
f0...f2	(2, 1, 0), (2, 2, 0), (0, 2, 0)
tx	(0, 1, 0)

Treatment D2: Here we modify treatment D1 by placing the f output neurons in a column, instead of surrounding a given point.

Neuron	Location in substrate
v0...v2	(0, 0..2, 0)
p0...p2	(0, 0..2, 1)
rx0...rx3	(2, 1, 2), (2, 2, 2), (0, 2, 2), (0, 1, 2)
f0...f2	(0, 0, 0), (0, 1, 0), (0, 2, 0)
tx	(0, 0, 1)

Treatment D3: In this treatment, we placed each different f output at position 1 in different axes. This was done to test if HyperNEAT was better able to isolate neurons that have only a single non-zero coordinate.

Neuron	Location in substrate
v0...v2	(0, 0..2, 0)
p0...p2	(0, 0..2, 1)
rx0...rx3	(2, 1, 2), (2, 2, 2), (0, 2, 2), (0, 1, 2)
f0...f2	(1, 0, 0), (0, 1, 0), (0, 0, 1)
tx	(0, 0, 0)

Treatment D4: To test if 3D coordinate frames are inherently better suited for the grid-coverage problem, here we placed different neuron types at different z coordinates.

Neuron	Location in substrate
v0...v2	(1, 0, 0), (0, 1, 0), (-1, 0, 0)
p0...p2	(1, 0, 1), (0, 1, 1), (-1, 0, 1)
rx0...rx3	(1, 0, 2), (0, 1, 2), (-1, 0, 2), (0, -1, 2)
f0...f2	(1, 0, 3), (0, 1, 3), (-1, 0, 3)
tx	(0, 0, 3)

Treatment D5: This treatment modified treatment D1 by adding a bias neuron to a single location within one layer of the substrate.

Neuron	Location in substrate
v0...v2	(0, 0..2, 0)
p0...p2	(0, 0..2, 1)
rx0...rx3	(2, 1, 2), (2, 2, 2), (0, 2, 2), (0, 1, 2)
f0...f2	(2, 1, 0), (2, 2, 0), (0, 2, 0)
tx	(0, 1, 0)
bias	(-1, -1, -1), layer 0 only.

Treatment F1: This treatment was an attempt to take into account all “common knowledge” of the HyperNEAT system (as given on the HyperNEAT FAQ: <http://eplex.cs.ucf.edu/hyperNEATpage/HyperNEAT.html>). Here, all coordinates are normalized to be in the range [0, 1] and a bias neuron was included.

Neuron	Location in substrate
v0...v2	(0.0..0.2, -1..1)
p0...p2	(0.3..0.5, -1..1)
rx0...rx3	(0.6..0.9, -1..1)
f0...f2	(0.0..0.5, 1)
tx	(0.75, 1)
bias	(-1, 0)

Treatment NE: Here, the substrate structure was the same as treatment H3, however the semantics of the rx sensors were changed from radio strength of sent messages to $1/\text{distance}$ to the nearest agents, without respect to their location.

Neuron	Location in substrate
v0...v2	(0..2, 0)
p0...p2	(0..2, 1)
rx0...rx3	(0..2, 2), (0, 3)
f0...f2	(0..2, 5)
tx	(-1, 6)

Treatment NH: This treatment was configured as treatment D3, but with no hidden layer.

Neuron	Location in substrate
v0...v2	(0, 0..2, 0)
p0...p2	(0, 0..2, 1)
rx0...rx3	(2, 1, 2), (2, 2, 2), (0, 2, 2), (0, 1, 2)
f0...f2	(2, 1, 0), (2, 2, 0), (0, 2, 0)
tx	(0, 1, 0)

Treatment T: Here, HyperNEAT CPPN was modified to return two weights, one for the input-hidden layer, and another for the hidden-output layer. This is listed as one of the options to try with HyperNEAT at: <http://eplex.cs.ucf.edu/hyperNEATpage/HyperNEAT.html>.

Neuron	Location in substrate
v0...v2	(0, 0..2)
p0...p2	(0, 3..5)
rx0...rx3	(1, 0..3)
f0...f2	(0, 0..2)
tx	(0, 3)

A.3 Multi-agent HyperNEAT-based Experiments

Here we describe the different Multi-agent HyperNEAT-based configurations that we tested on the grid-coverage problem. In general, the substrate structures used here were based on those from the HyperNEAT treatments, with alterations made as needed to suit the Multi-agent HyperNEAT algorithm.

Treatment R1: This treatment used the same substrate structure as treatment H1, however each edge weight was produced as the result of five inputs to the CPPN. Specifically, both global and local x and y coordinates were used, in addition to l (the layer index).

Neuron	Location in substrate
v0...v2	(0, 0..2)
p0...p2	(0, 3..5)
rx0...rx3	(1, 0..3)
f0...f2	(0, 0..2)
tx	(0, 3)

Treatment R2: This treatment was configured similarly to treatment R1, though it used the substrate structure from treatment D2.

Neuron	Location in substrate
v0...v2	(0, 0..2, 0)
p0...p2	(0, 0..2, 1)
rx0...rx3	(2, 1, 2), (2, 2, 2), (0, 2, 2), (0, 1, 2)
f0...f2	(0, 0, 0), (0, 1, 0), (0, 2, 0)
tx	(0, 0, 1)

Treatment M1: This treatment used the same substrate structure as treatment H1, however edge weights were produced as the result of four inputs to the CPPN: global x and y coordinates, agent index, and layer index within the substrate.

Neuron	Location in substrate
v0...v2	(0, 0..2)
p0...p2	(0, 3..5)
rx0...rx3	(1, 0..3)
f0...f2	(0, 0..2)
tx	(0, 3)

Treatment M2: This treatment used the same substrate structure as treatment D1, and was otherwise configured as treatment M1.

Neuron	Location in substrate
v0...v2	(0, 0..2, 0)
p0...p2	(0, 0..2, 1)
rx0...rx3	(2, 1, 2), (2, 2, 2), (0, 2, 2), (0, 1, 2)
f0...f2	(2, 1, 0), (2, 2, 0), (0, 2, 0)
tx	(0, 1, 0)

Treatment M3: This treatment was configured as treatment M1, with the addition of a bias neuron.

Neuron	Location in substrate
v0...v2	(0, 0..2)
p0...p2	(0, 3..5)
rx0...rx3	(1, 0..3)
f0...f2	(0, 0..2)
tx	(0, 3)
bias	(-1, -1), layer 0 only.

Treatment F2: This treatment attempted to take into account all “common knowledge” as given by the HyperNEAT FAQ: <http://eplex.cs.ucf.edu/hyperNEATpage/HyperNEAT.html>.

Neuron	Location in substrate
v0...v2	(0.0..0.2, -1..1)
p0...p2	(0.3..0.5, -1..1)
rx0...rx3	(0.6..0.9, -1..1)
f0...f2	(0.0..0.5, 1)
tx	(0.75, 1)
bias	(-1, 0)