18th ICCRTS

Learned Tactics for Asset Allocation

Topics

Topic 5: Experimentation, Metrics, and Analysis
Topic 6: Modeling and Simulation
Topic 3: Data, Information and Knowledge

Authors

David B. D'Ambrosio
Douglas S. Lange
Space and Naval Warfare Systems Center - Pacific
San Diego, CA 92152

Point of Contact

Douglas S. Lange
Space and Naval Warfare Systems Center - Pacific
(619)553-6534
doug.lange@navy.mil

Abstract

Tactics can be developed in a number of different ways. Rules can be created based on a theory of operations as has been done in the development of tactical decision aids for a considerable time. But these tools can behave poorly in unanticipated scenarios and can require significant design effort. In this paper, an existing machine learning approach for training geographically based agents learns tactics for the placement of surveillance assets. These results serve as a potential benchmark to compare against other methods. While there are many approaches possible, there are currently few ways of directly comparing these methods for the military environment. It would be advantageous to have a common set of benchmark scenarios that could evaluate different strategies with respect to each other. This paper presents such a problem, simulated data, and solution. In this domain a limited number of surveillance assets must autonomously coordinate to detect several types of vessels, so that they can be intercepted. The results show that a machine learning approach is able to consistently locate opposing vessels, even in the presence of noise, but more importantly provides a performance baseline and guide for developing future benchmark problems.

1 Introduction

Determining tactics and plans for command and control (C2) is a common problem. Traditionally, these tactics were devised by human experts after careful consideration of available intelligence. However, the increase of available information and demand for more complex tactics necessitates the use of computational resources in the process. Such approaches typically rely on defined rules and conditions that can automatically determine the appropriate response to a situation or provide guidance for humans. But the process of defining these criteria is time consuming and can have poor results when applied to unanticipated scenarios. Thus, machine learning (ML), an artificial intelligence approach that builds systems by examining data, is a rapidly growing and effective means of generating robust tactics to a variety of C2 problems.

While there are many benefits of applying ML to C2, there are also many possible ways to do so. Therefore a common question is "What is the best machine learning approach for this problem?" Unfortunately, there is currently no simple means of easily determining the best approach without resorting to implementing and trying out several methods, which can require significant time and monetary investment. Thus, it would be advantageous to have benchmark tasks that could give a general idea of the effectiveness of various machine learning approaches across various classes of domains.

In this paper, such a benchmark task is presented in the form of a surveillance domain in which two aerial surveillance planes must detect drug runners operating in the waters around central America. This problem provides an interesting test bed for machine learning because the optimal solution is not necessarily known, and, in fact, may change with different circumstances. Additionally, these planes may need to operate with degraded or limited communication, which can be challenging to many machine learning algorithms. However, it is easy to judge the quality of solutions based on how many drug runners are spotted. To solve this problem, the machine learning technique multiagent HyperNEAT is employed to determine the tactics for the two planes, with the assumption that there is no communication between them. The results show that the planes are able to detect significantly more boats than hand-coded heuristics, and provides the opportunity for comparison with other machine learning approaches.

2 Background

This section reviews relevant benchmarking approaches, C2 for asset allocation (including multiagent approaches), and the NEAT and HyperNEAT methods that form the backbone of the multiagent HyperNEAT approach.

2.1 Benchmarking

Ranking things is an almost fundamental human desire and it can be very important when trying to select among a group of competing approaches for a particular application. However, it has been notoriously difficult to measure the effectiveness of C2 systems [53] and there are currently few effective means to do so. Interestingly, the increase of machine learning and other computational approaches to C2 bring with them a number of benchmarking techniques from those fields. For example, there are a number of benchmark problems in the machine learning community where all approaches try and learn the most information from the same set of data [33] such as handwriting recognition [20] or diagnosing diseases from patient data [35]. Such benchmarks allow the demonstration of improvement upon or against an existing algorithm. There are dangers of over-fitting to the benchmark, so it is important that the benchmark problem shares a relationship with (or even better, actually is) a real-world problem to be solved. It is also true that techniques will do better on some benchmarks than others, but that can actually be helpful in determining which types of problems that technique is best suited to solve. The benchmark domain proposed in this paper is such a real-world problem and should improve the ability to judge the effectiveness of a C2 approach.

2.2 C2 for Asset Allocation

Aside from human-determined strategies, there are a number of algorithm and machine learning approaches to C2 of assets. One popular technique is the application of game theory [43]. In these cases, the scenario requiring C2 is modeled as an adversarial game and Nash equilibrium states (i.e. all players cannot do anything differently without one of them losing points) can be solved for. However, the rules for different games are very strict and thus may not represent real-world scenarios accurately. Other approaches involve defining sets of states and transitions known as finite state automata [36], but when unanticipated states are encountered, there may not be an adequate response defined. Multiagent Learning, discussed in the next section, is a promising approach to asset allocation and control that is applied in this paper.

2.3 Cooperative Multiagent Learning

When multiple assets must be commanded, it can be beneficial to model the problem as a multiagent system. That is, assets and other important elements of the domain are modeled as interacting agents, which allows for complex simulations and experiments that can model various outcomes. Multiagent systems confront a broad range of domains, creating the opportunity for real-world applications such as room clearing, pursuit [15], and synchronized motion [45]. 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.

There are two primary traditional approaches to multiagent learning. The first, multiagent reinforcement learning (MARL), encompasses several specific techniques based on off-policy and on-policy temporal difference learning [6, 31, 47]. The basic principle that unifies MARL techniques is to identify and reward promising cooperative states and actions among a team of agents [7, 40]. The other major approach, cooperative coevolutionary algorithms (CCEAs), is an established evolutionary method for training teams of agents that must work together [18, 40, 41]. 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.

While reinforcement learning and evolution are mainly the focus of separate communities, Panait, Tuyls, and Luke [42] showed 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 appealing, one problem is that when individual roles are learned separately, there is no representation of how roles relate to the team structure and therefore no principle for exploiting regularities that might be shared across all or part of the team. Thus in cases where learning has been applied to real-world applications, it usually exploits inherent homogeneity in the task [13, 44]. The approach in this paper, multiagent HyperNEAT addresses many of these issues.

2.4 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) [19, 67]. 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 [46, 52]. Early attempts at NE used fixed-topology models that were designed by the experimenter [39]. 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 [30, 32, 55]. 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.

2.4.1 NeuroEvolution of Augmenting Topologies (NEAT)

The NEAT method was originally developed to evolve ANNs to solve difficult control and sequential decision tasks [55, 57, 59], the basic principles of NEAT are reviewed in this section.

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 [25, 48], or arbitrary random-topology networks [3, 26, 67]. 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 [37, 65] and shown to improve adaptation in a few prior evolutionary [2] and neuroevolutionary [28] 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, 58, 60, 62]. Complete descriptions of the NEAT method, including experiments confirming the contributions of its components, are available in Stanley and Miikkulainen [55, 57] and Stanley et al. [59].

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 [26, 67]. 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.

2.4.2 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 [4, 34, 56]. 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* [4, 5, 17, 29, 34, 38, 51, 54, 56].

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 [21–24], adaptive maze navigation [46], quadruped locomotion [11], keepaway soccer [63, 64] and a variety of others [8–10, 12, 16, 27, 61, 66]. For a full description of HyperNEAT see Stanley et al. [61] and Gauci and Stanley [24].

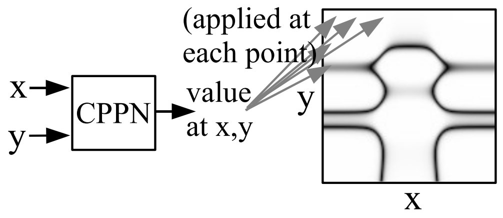
In HyperNEAT, NEAT is altered to evolve an indirect encoding called compositional pattern producing networks (CPPNs [54]) *instead* of ANNs. CPPNs are a high-level abstraction of the development process in nature, intended to approximate its representational power without the computational cost. The 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 [54].

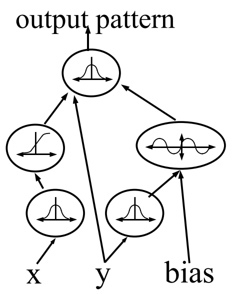
Specifically, CPPNs produce a phenotype that is a function of n dimensions, where n is the number of dimensions of the desired solution, for example, $n = 2$ for a two-dimensional image. For each coordinate in that space, its level of expression is output by the CPPN, which encodes the phenotype. Figure 1 shows how a two-dimensional phenotype can be generated by a function of two parameters (x and y) that is represented by a network of composed functions that produce intensitiy values for each set of parameters. The CPPN in figure 1b actually produces the pattern in a. Because CPPNs are a superset of traditional ANNs, which can approximate any function [14], CPPNs are also universal function approximators. Thus a CPPN can encode any pattern within its n-dimensional space.

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 [49, 50, 54]. 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 [54] including an online service on which users collaboratively breed patterns represented by CPPNs [49, 50].

The main idea in HyperNEAT is that CPPNs can also naturally encode *connectivity patterns* [21, 22, 24, 61, 64]. 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



(a) Pattern Encoding



(b) CPPN

Figure 1: CPPN Encoding

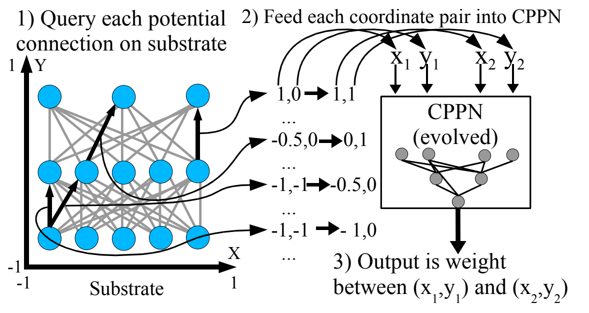

1) Query each potential connection on substrate
2) Feed each coordinate pair into CPPN
1 Y
-1
-1
X
1
Substrate
1,0 → 1,1
-0.5,0 → 0,1
-1,-1 → -0.5,0
-1,-1 → - 1,0
x1 y1 x2 y2
CPPN (evolved)
3) Output is weight between (x1,y1) and (x2,y2)

Figure 2: Hypercube-based Geometric Connectivity Pattern Interpretation

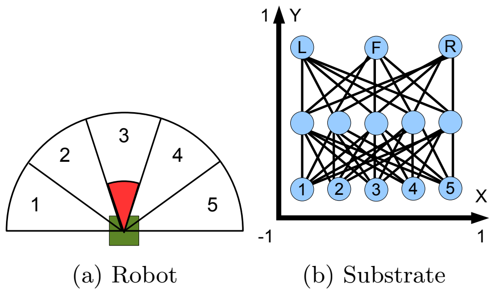


Figure 3: Substrate Configuration

policies can be produced across possible agent locations. The key insight in HyperNEAT is that $2n$-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 (Figure 2). 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 [10, 11, 22, 24, 61, 64]. That way, the connectivity of the substrate is a function of the the task structure.

For example, the sensors of an autonomous robot can be placed from left to right on the substrate in the same order that they exist on the robot (Figure 3). Outputs for moving left or right can also be placed in the same order, implying a relationship between the sensors and effectors. Such placement allows the CPPN to generate connectivity patterns easily that respect the geometry of the problem, such as left-right symmetry. 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.

3 Approach

The approach applied in this paper combines two existing extensions to the HyperNEAT method: HyperNEAT-BEV and multiagent HyperNEAT, both of which are described in this section.

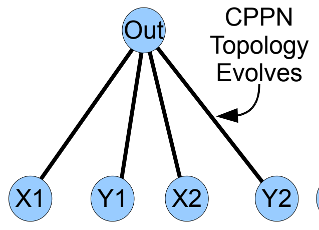
3.1 HyperNEAT-BEV

The Bird's Eye View (BEV) extension to HyperNEAT allows it to accept complex information as input represented as a static overhead view, which is a a common approach for visualizing and understanding C2 domains for humans. This extension was applied by Verbancsics and Stanley [64] in several strategic decision domains such as Robocup soccer. This approach is advantageous because it allows HyperNEAT to exploit patterns in the information as a whole, rather than piecewise. Inputting information in this manner is also beneficial because the state-space remains fixed rather than changing egocentrically.

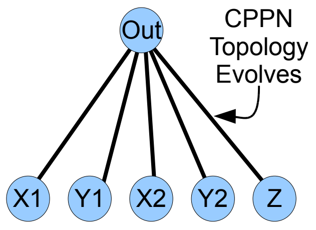
In previous applications of BEV, while there were multiple agents present, only one would act per time step. In this paper, multiple agents must act simultaneously, and thus multiple outputs are required. Therefore the multiagent HyperNEAT paradigm is employed to incentivize cooperation and coordination among their behaviors.

3.2 Multiagent HyperNEAT

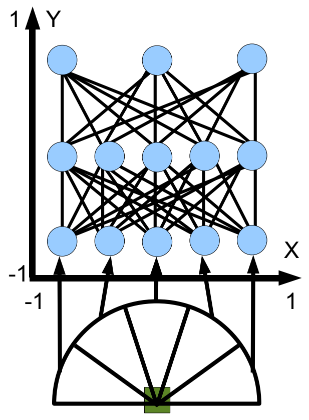
Multiagent HyperNEAT is based on the idea that a team of cooperating agents can be defined by describing the relationship of policies to each other (referred to as the team's *policy geometry*. To understand how the policy geometry of a team can be encoded, it helps to begin by considering *homogeneous 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.



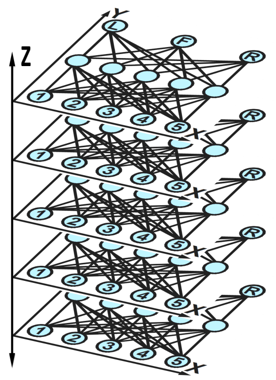
(a) Homogeneous CPPN



(b) Heterogeneous CPPN



(c) Homogeneous Substrate



(d) Heterogeneous Substrate

Figure 4: Multiagent HyperNEAT Encoding

3.2.1 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 (Figure 4a) queries the substrate shown in Figure 4c, which has five inputs, five hidden nodes, and three output nodes, to determine its connection weights. This substrate is designed to 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 [61] 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.

3.2.2 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 (Figure 4b,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.

The heterogeneous substrate (Figure 4d) 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.

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.

4 Surveillance Experiment

In this domain, two planes (P-3 AIPs) must patrol the waters of Central America with the goal of spotting boats that are smuggling narcotics over a 72 hour period. The area that must be searched is vast and separated by land, so the planes must cooperate to effectively cover it. Of benefit to the planes, however, is that there is some degree of information about the smugglers, although much of it is uncertain. HyperNEAT is a natural fit for this problem because of the large input space and cooperative nature of the problem. There are two agent types to be considered: smugglers and searchers.

Planning asset allocation depends largely on intelligence gathered. In this scenario, the important information known about the smuggler boats is the path they will take, the time they will depart, and the speed at which they will travel. The paths all begin in South America, lead to a point in Central America, and are defined by one to four waypoints that the boats will travel through. It is assumed that the boats will take the most direct route to each waypoint. Both the waypoints and the departure time have a known uncertainty value, which are used to generate probability distribution heatmaps that represent the chance that a smuggler boat will be at a given location for given time, which is dependent on the uncertainty associated with the intelligence related to the boats. Figure 5 shows an example heatmap with multiple possible boat tracks. In this experiment, 72 (one for each hour of the scenario) such distributions are created at a 1 degree by 1 degree resolution and serve as the basis for deciding the patrol routes of the searchers.

The searcher planes both start at the same location (89,13) and can move up to 325 nautical miles every hour. To determine the patrol route the planes will take, a substrate is defined that takes the probability heatmaps defined above as input at $z = 0$. The outputs are two sheets of neurons at $z = 1$ and -1 that

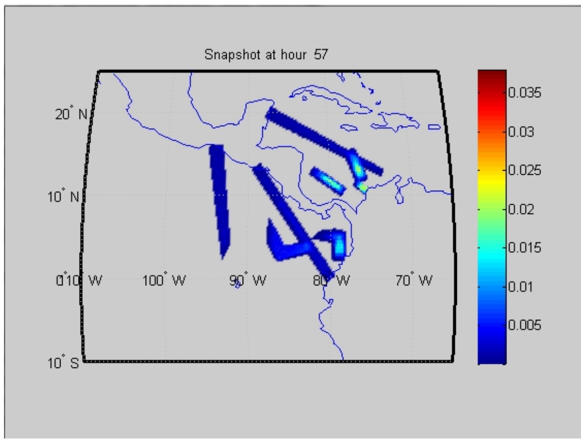


Figure 5: Heatmap Example

are the same size as the heatmaps above and represent a desired location to move to (figure 6). The substrate takes as input a heatmap at $z = 0$, which is fully connected to two equally sized output maps at $z = -1$ and $z = 1$. This substrate construction is slightly different than the traditional multiagent HyperNEAT approach in section 3.2, and would likely require additional input dimension for more than two planes. The highest output neuron within a plane's movement radius is where the plane will move in the next hour. This process is repeated for each timestep to produce the patrol route for both planes. A smuggler boat is considered spotted if it and a plane occupy the same cell at the same hour. Importantly, this approach does not assume communication between the planes, that is the policy of one plane does not depend on the actions of another plane; they are both defined by the substrate. Thus, environments with limited, intermittent, or even non-existent communication could benefit from a multiagent HyperNEAT approach, because it allows the agents to effectively cooperate without explicit communication.

To evaluate the patrols, 25 Monte Carlo simulations are run and assigned a fitness based on the number of boats spotted and how close the plane was to spotting the unspotted boats. Thus, $\frac{1}{B}$, where B is the number of boats is added to the fitness for each spotted boat and $0.1 \times 1 - \frac{d_b}{d_max} \times \frac{1}{B}$ where d_b is the

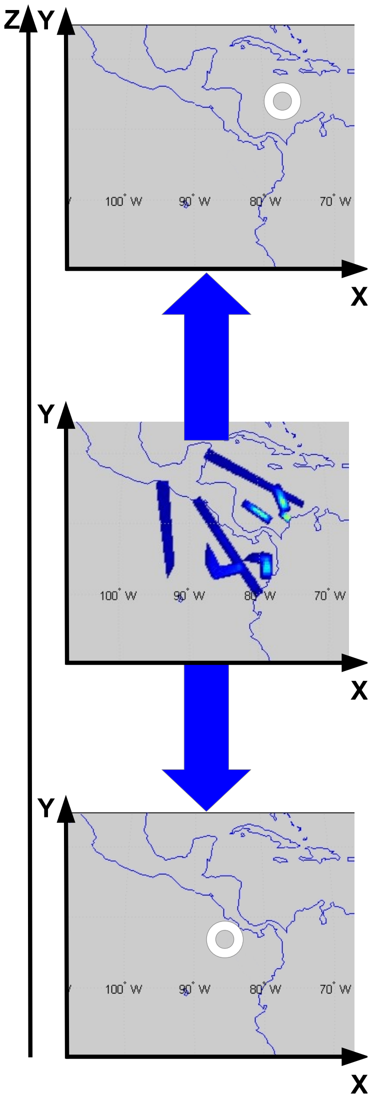


Figure 6: Neural Network Substrate

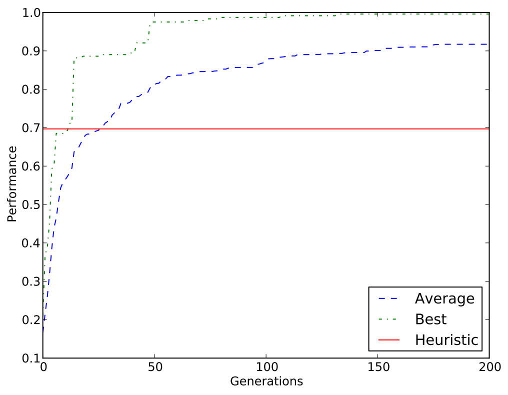


Figure 7: Training Results

closest a plane got to the given boat and d_max is the maximum distance a boat could be from a plane for each unspotted boat. This approach allows for a smoother search gradient by partially rewarding strategies that come close to discovering smugglers. Note that the input heatmaps are pre-generated and not affected by the actual behavior of the boats during the simulation. Thus the planes must come up with patrol routes that will work in all situations.

5 Results

The results show that, on average, strategies that spot 90% of boats are found within 149 generations, although the best run found such solutions in only 43 generations (figure 7). When compared to the hand-coded strategy of moving towards the area of highest probability, while minimizing coverage overlap, evolved solutions are, on average, significantly better within 26 generations ($p < 0.05$ by Student's t-test). To test generalization, the best strategies from each generation were further evaluated on 250 Monte Carlo simulations that had not been seen in training and the average performance was not significantly different ($p < 0.05$) than training.

6 Discussion and Future Work

The results show that multiagent HyperNEAT can find effective patrol routes for this domain. Qualitatively, different evolutionary runs produced several different types of solutions. The two planes would always initially split up, but from there several tactics were employed. Some planes found high traffic locations and stayed in those general areas, whereas other planes moved around the entire map frequently. Imposing additional goals or costs (e.g. fuel expenditure, coverage, etc.) could shift the ideal policy closer to one of these extremes, but the fact that there are multiple effective solutions in this general case demonstrates the utility of evolutionary approaches in finding "creative" solutions to problems.

There are several future research directions based on this work. Firstly, the C2 problem could be extended to include additional agent types such as friendly boats that will actually intercept detected smugglers. An alternative extension would be to increase the accuracy of the solutions by inputting and outputting higher resolution heatmaps and scaling up the substrate, as has been done in many other HyperNEAT problems [24]. However, the ANNs in this paper were already quite large (over 1.5 million connections), so training larger networks could be a time consuming. Finally, applying other C2 machine learning approaches and comparing them to the HyperNEAT results would provide interesting benchmarking opportunities.

7 Conclusion

This paper presented a relevant C2 domain in the form of determining patrol routes for planes searching for drug smugglers and a machine learning approach to solving it. The approach, multiagent HyperNEAT, was able to find multiple effective routes that significantly outperformed hand-coded heuristics. Ultimately this domain can serve as a new benchmark domain for comparing C2 approaches in the future.

8 Acknowledgments

Thanks to Michael Atkinson for providing initial heatmap data and visualization.

References

[1] Aaltonen et al. (over 100 authors). Measurement of the top quark mass with dilepton events selected using neuroevolution at CDF. *Physical Review Letters*, 102(15):2001, 2009.

[2] L. Altenberg. Evolving better representations through selective genome growth. In *Proceedings of the IEEE World Congress on Computational Intelligence*, pages 182–187, Piscataway, NJ, 1994. IEEE Press.

[3] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5:54–65, 1993.

[4] P. J. Bentley and S. Kumar. The ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, pages 35–43, San Francisco, 1999. Kaufmann.

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

[6] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

[7] L. Busoniu, B. D. Schutter, and R. Babuska. Learning and coordination in dynamic multiagent systems. Technical Report 05-019, Delft University of Technology, 2005.

[8] J. Clune, C. Ofria, and R. Pennock. How a generative encoding fares as problem-regularity decreases. In *Proceedings of the 10th International Conference on Parallel Problem Solving From Nature (PPSN 2008)*, pages 258–367, Berlin, 2008. Springer.

[9] J. Clune, B. B. Beckmann, R. Pennock, and C. Ofria. HybrID: A hybridization of indirect and direct encodings for evolutionary computation. In *Proceedings of the European Conference on Artificial Life (ECAL-2009),*, 2009.

[10] J. Clune, B. E. Beckmann, C. Ofria, and R. T. Pennock. 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*, Piscataway, NJ, USA, 2009. IEEE Press.

[11] J. Clune, R. T. Pennock, and C. Ofria. The sensitivity of HyperNEAT to different geometric representations of a problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2009)*, New York, NY, USA, 2009. ACM Press.

[12] J. Clune, B. Beckmann, P. McKinley, and C. Ofria. Investigating whether HyperNEAT produces modular neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2010)*, New York, NY, 2010. ACM Press. URL `http://eplex.cs.ucf.edu/publications/2010/verbancsics.gecco10.html`.

[13] N. Correll and A. Martinoli. Collective inspection of regular structures using a swarm of miniature robots. In M. Ang and O. Khatib, editors, *Experimental Robotics IX*, volume 21 of *Springer Tracts in Advanced Robotics*, pages 375–386. Springer Berlin / Heidelberg, 2006.

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

[15] D. B. D'Ambrosio, J. Lehman, S. Risi, and K. O. Stanley. Evolving policy geometry for scalable multiagent learning. In *Proceedings of the 9th International Conference on Autonomous Agents*

and Multiagent Systems: volume 1-Volume 1, pages 731–738. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[16] J. Drchal, J. Koutnk, and M. Snorek. HyperNEAT controlled robots learn to drive on roads in simulated environment. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2009)*, Piscataway, NJ, USA, 2009. IEEE Press.

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

[18] S. Ficici and J. Pollack. A game-theoretic approach to the simple coevolutionary algorithm. *Lecture notes in computer science*, pages 467–476, 2000.

[19] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1:47–62, 2008.

[20] P. Frey and D. Slate. Letter recognition using holland-style adaptive classifiers. *Machine Learning*, 6(2):161–182, 1991.

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

[22] J. Gauci and K. O. Stanley. A case study on the critical role of geometric regularity in machine learning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008)*, Menlo Park, CA, 2008. AAAI Press. URL `http://eplex.cs.ucf.edu/papers/gauci_aaai08.pdf`.

[23] J. Gauci and K. O. Stanley. Indirect encoding of neural networks for scalable go. In R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph, editors, *Parallel Problem Solving from Nature – PPSN XI*, volume 6238 of *Lecture Notes in Computer Science*, pages 354–363. Springer, 2010. ISBN 978-3-642-15843-8.

[24] J. Gauci and K. O. Stanley. Autonomous evolution of topographic regularities in artificial neural networks. *Neural Computation*, 22(7):1860–1898, 2010. URL `http://eplex.cs.ucf.edu/publications/2010/gauci.nc10.html`.

[25] F. Gomez and R. Miikkulainen. Solving non-Markovian control tasks with neuroevolution. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 1356–1361, San Francisco, 1999. Kaufmann. URL `http://nn.cs.utexas.edu/keyword?gomez:ijcai99`.

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

[27] E. Haasdijk, A. Rusu, and A. Eiben. HyperNEAT for Locomotion Control in Modular Robots. *Evolvable Systems: From Biology to Hardware*, pages 169–180, 2010.

[28] I. Harvey. *The Artificial Evolution of Adaptive Behavior*. PhD thesis, School of Cognitive and Computing Sciences, University of Sussex, Sussex, 1993. URL `http://www.cogs.susx.ac.uk/users/inmanh/inman_thesis.html`.

[29] G. S. Hornby and J. B. Pollack. Creating high-level components with a generative representation for body-brain evolution. *Artificial Life*, 8(3), 2002. URL `http://www.demo.cs.brandeis.edu/papers/hornby_alife02.pdf`.

[30] P. Hotz, G. Gomez, and R. Pfeifer. 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 andSynthesis of Living Systems*, volume 2003, 2003.

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

[32] J. R. Koza and J. P. Rice. Genetic generalization of both the weights and architecture for a neural network. In *Proceedings of the International Joint Conference on Neural Networks* (New York, NY), volume 2, pages 397–404, Piscataway, NJ, 1991. IEEE.

[33] F. Leisch and E. Dimitriadou. Machine learning benchmark problems. 2010.

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

[35] O. Mangasarian and W. Wolberg. *Cancer diagnosis via linear programming.* University of Wisconsin-Madison, Computer Sciences Department, 1990.

[36] A. Marino, L. Parker, G. Antonelli, and F. Caccavale. Behavioral control for multi-robot perimeter patrol: A finite state automata approach. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 831–836. IEEE, 2009.

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

[38] J. F. Miller. Evolving a self-repairing, self-regulating, French flag organism. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, Berlin, 2004. Springer Verlag. URL `http://www.elec.york.ac.uk/intsys/users/jfm7/gecco2004.pdf`.

[39] D. J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 762–767. San Francisco: Kaufmann, 1989.

[40] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 3(11):383–434, November 2005. ISSN 1573-7454. doi: 10.1007/s10458-005-2631-2. URL `http://jmvidal.cse.sc.edu/library/panait05a.pdf`.

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

[42] L. Panait, K. Tuyls, and S. Luke. Theoretical Advantages of Lenient Learners: An Evolutionary Game Theoretic Perspective. *The Journal of Machine Learning Research*, 9:423–457, 2008.

[43] J. Pita, M. Jain, J. Marecki, F. Ordóñez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus. Deployed armor protection: the application of a game theoretic model for security at the los angeles international airport. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, pages 125–132. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[44] M. Quinn, L. Smith, G. Mayley, and P. Husbands. Evolving controllers for a homogeneous system of physical robots: Structured cooperation with minimal sensors. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 361(1811):2321, 2003. ISSN 1364-503X.

[45] M. Quinn, L. Smith, G. Mayley, P. Husbands, M. Quinn, L. Smith, G. Mayley, and P. Husbands. Evolving controllers for a homogeneous system of physical robots: Structured cooperation with minimal sensors. *Philosophical Transactions of the Royal Society of London. Series*

A: Mathematical, Physical and Engineering Sciences, 361(1811):2321–2343, 2003.

[46] S. Risi and K. O. Stanley. Indirectly encoding neural plasticity as a pattern of local rules. In *Proceedings of the 11th International Conference on Simulation of Adaptive Behavior (SAB2010)*, Berlin, 2010. Springer.

[47] H. Santana, G. Ramalho, V. Corruble, and B. Ratitch. Multi-agent patrolling with reinforcement learning. *Autonomous Agents and Multiagent Systems, International Joint Conference on*, 3:1122–1129, 2004.

[48] N. Saravanan and D. B. Fogel. Evolving neural control systems. *IEEE Expert*, pages 23–27, June 1995.

[49] J. Secretan, N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, and K. O. Stanley. Picbreeder: Evolving pictures collaboratively online. In *CHI '08: Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1759–1768, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-011-1. doi: http://doi.acm.org/10.1145/1357054.1357328.

[50] J. Secretan, N. Beato, D. B. D.Ambrosio, A. Rodriguez, A. Campbell, J. T. Folsom-Kovarik, and K. O. Stanley. Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evolutionary Computation*, 2011. To appear.

[51] K. Sims. Evolving 3D morphology and behavior by competition. In R. A. Brooks and P. Maes, editors, *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems (Artificial Life IV)*, pages 28–39. MIT Press, Cambridge, MA, 1994. URL `http://www.mpi-sb.mpg.de/services/library/proceedings/contents/alife94.html`.

[52] A. Soltoggio, A. J. Bullinaria, C. Mattiussi, P. Dürr, and D. Floreano. Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. In S. Bullock, J. Noble, R. Watson, and M. Bedau, editors, *Proceedings of the Eleventh International Conference on Artificial Life (Alife XI)*, Cambridge, MA, 2008. MIT Press. URL `http://eplex.cs.ucf.edu/papers/lehman_alife08.pdf`.

[53] N. Sproles. The difficult problem of establishing measures of effectiveness for command and control: A systems engineering perspective. *Systems engineering*, 4(2):145–155, 2001.

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

[55] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002. URL `http://nn.cs.utexas.edu/keyword?stanley:ec02`.

[56] K. O. Stanley and R. Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9 (2):93–130, 2003. URL `http://nn.cs.utexas.edu/keyword?stanley:alife03`.

[57] K. O. Stanley and R. Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100, 2004. URL `http://nn.cs.utexas.edu/keyword?stanley:jair04`.

[58] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Evolving neural network agents in the NERO video game. In *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games*, 2005.

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

[60] K. O. Stanley, N. Kohl, and R. Miikkulainen. Neuroevolution of an automobile crash

warning system. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2005. URL `http://nn.cs.utexas.edu/keyword?stanley:gecco05`.

[61] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based indirect encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009. URL `http://eplex.cs.ucf.edu/publications/2009/stanley.alife09.html`.

[62] M. E. Taylor, S. Whiteson, and P. Stone. Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In *GECCO 2006: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1321–1328, July 2006.

[63] P. Verbancsics and K. O. Stanley. Task transfer through indirect encoding. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)*, New York, NY, 2010. ACM Press. URL `http://eplex.cs.ucf.edu/publications/2010/verbancsics.gecco10.html`.

[64] P. Verbancsics and K. O. Stanley. Evolving static representations for task transfer. *Journal of Machine Learning Research (JMLR)*, 11: 1737–1769, 2010.

[65] J. D. Watson, N. H. Hopkins, J. W. Roberts, J. A. Steitz, and A. M. Weiner. *Molecular Biology of the Gene Fourth Edition*. The Benjamin Cummings Publishing Company, Inc., Menlo Park, CA, 1987.

[66] B. G. Woolley and K. O. Stanley. Evolving a single scalable controller for an octopus arm with a variable number of segments. In R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph, editors, *Parallel Problem Solving from Nature – PPSN XI*, volume 6239 of *Lecture Notes in Computer Science*, pages 270–279. Springer, 2010. ISBN 978-3-642-15870-4.

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