An Investigation of Neuroevolution Methods for Evolving Multi-Robot Team Controllers to Solve Complex Collective Behaviour Tasks

A. Ruben Putter
University of Cape Town
Rondebosch 7701
Cape Town, South Africa
PTTAND010@myuct.ac.za

ABSTRACT

Neuroevolution uses Evolutionary Algorithms (EAs) to evolve connection weights and network topologies of Artificial Neural Networks (ANNs) in order to learn a specific task. This literature review investigates different NE methods and their effectiveness in solving complex team tasks. Objective, novelty and hybrid searches are also examined as alternatives to the standard fitness function. It outlines the basics of ANNs such as the different training methods and topologies as well as compares direct and indirect encoding techniques for NE.

Keywords

HyperNEAT; Neuroevolution; Artificial Neuron Networks; Evolutionary Algorithms; Indirect Encoding; Direct Encoding

1. INTRODUCTION

Evolutionary computing (EC) is a branch of computer science research inspired by the Darwinian process of evolution through natural selection[7]. Given the complexity and non-linearity of real world problems, traditional controller design is impractical since they rely on linear mathematical models [16], making it unsurprising that computer scientists have turned to natural processes for inspiration. The problem solving power of evolution is abundantly clear in nature with the diverse range of organisms on Earth, each specially designed for optimal survival in its own niche environment [7].

Although Reinforcement Learning (RL) techniques can theoretically solve a number of problems without examples of correct behaviour, in practice they scale poorly with problems that have large state spaces or non-Markov tasks, which are tasks where the state of the environment is only partially observable [16].

Neuroevolution (NE) is a method of artificially evolving Artificial Neural Networks (ANNs) by implementing Evolutionary Algorithms (EAs) to perform tasks such as training the connection weights and designing the network topology[11, 45].

By evolving a population of ANNs, NE searches through the space of possible controllers by evaluating each candidate solution using a fitness function and selecting the most successful individuals for reproduction, a process analogous to natural selection [16, 14].

The different NE methods can be roughly split into two categories according to their encoding method, either direct or indirect. In direct encoding methods, each element in the genotype explicitly encodes an independent aspect of the phenotype, such that there is a one-to-one mapping between the genotype and the phenotype. In indirect (also called generative or developmental) encoding methods, each element in the genotype is implicitly described by a computable function allowing for a much more compact representation of the genotype. It also allows for genetic information to be reused [3, 40].

The high level of complexity inherent in real life tasks tend to result in deceptive fitness landscapes [13]. These are fitness landscapes of multimodal problems that have numerous local optima [7]. In multimodal problems, it is possible for the NE search to get stuck around a local optima and lead to the NE method prematurely converging to a suboptimal solution [13].

For this reason, various NE search functions (also referred to as fitness functions) are investigated. The first search function being the objective search which simply evaluates the suitability of a solution by using an objective fitness function [28]. Novelty search is an approach that provides candidate solutions with a reward based how different their behaviour is from other candidate solutions [13]. The third approach is a hybrid search which attempts to combine EAs with a local search [1].

The collective construction task requires that a team of agents cooperate by coordinating their behaviours in order to assemble various structures within their environment [36].

The collective gathering task requires a team of agents to coordinate sub-tasks amongst themselves in order to produce a collective behaviour that maximises the resources gathered [36].

It has been shown that NE can successfully be applied to complex control tasks that require collective behaviour. These control tasks have no obvious mapping between the input from sensory configurations and motor outputs [36].

The ability to automate construction tasks through evolving ANN controllers for multi-robot teams using EAs presents several considerable benefits. Automation such as this could be used to assist in projects such as producing low-cost housing and reduce high accident rates due to human error that accompany traditional construction methods [23]. It would be useful to send multi-robot construction teams to extraterrestrial planets in order to build habitable structures in an-

ticipation of humans arriving. These robot teams would also be useful in underwater construction which is difficult and often extremely dangerous [44].

This literature review is therefore focussed on investigating the effectiveness of various NE methods in solving collective behaviour tasks, with specific emphasis on the indirect encoding method called HyperNEAT [39], as well as the aforementioned search functions.

Another reason for using HyperNEAT is its success in evolving team (collective) behaviours for various multi-agent tasks including RoboCup soccer and pursuit evasion [19].

2. NEUROEVOLUTION

Neuroevolution (NE) provides a way of combining EAs and ANNs [11]. It uses EAs to evolve the connection weights, topologies and activation functions of ANNs in order to learn a specific task or behaviour [15]. NE searches for an ANN with optimal performance based on a fitness function that determines an ANN's suitability to performing a task [11].

Using NE to evolve ANNs addresses several weaknesses that occur in reinforcement or supervised learning techniques. Reinforcement learning (RL) requires a value function which is costly to compute whereas NE removes the need of such a function by directly searching the policy space using a EA [16].

By searching through a population of solutions and evaluating several candidate solutions at a time, EAs are much less likely to get stuck in local minima than gradient-descent algorithms and therefore makes NE suitable for dealing with complex problems that generate numerous local optima [14, 45].

In a standard RL scenario an ANN interacts with its environment in discrete time steps [21]. NE can be used in any type of environment, whether it is continuous or partially observable, making it possible to find an optimal ANN using only information about how well the network is performing rather than what it is supposed to be doing [31].

Another challenge in using traditional learning methods or fixed-topology NE approaches, is that it requires a human to design the topology for the neural net. This presents a problem since these networks can get complex and would have to be altered according to the specific task being learned. This relation between topology and suitability is very unintuitive and difficult to get right without trial-and-error. By using a NE approach that evolves the topology and the weights of a neural network (TWEANNs), these networks are able to discover the most efficient topology [11].

Since the chromosomes in NE can be used to encode any aspect of ANNs and the choice of encoding affects the search space, deciding on an encoding method is fundamental aspect in the design of the system [11]. This literature review compares direct and indirect encoding, which will be discussed in depth at a later stage in the Encoding Schemes.

2.1 Artificial Neural Networks

Artificial Neural Networks (ANN's) are a simplified model of how a biological brain functions[30]. It consists of numerous simple computational units, called neurons, that are interconnected and ordered into layers to create a neural net[11]. Each neuron has several inputs, some activation function and a single output. An ANN receives input from the environment at its input layer [16].

Each neuron in a non-input layer then calculates the weighted sum of its inputs, referred to as the activation value [46, 11], and evaluates it according to some activation function. If this result exceeds a predetermined threshold value, the neuron will "fire" an output signal, transmitting it as an input to the subsequent neuron across a weighted connection [46].

ANNs are universal function approximators. It has been shown that a network can approximate any continuous function to any desired accuracy [47]. This makes ANNs a favourable choice for agent controllers in accomplishing various tasks [46].

In cases where training examples are available, the connection weights in ANNs are adapted using various supervised learning techniques, and in cases without such examples the weights are evolved using EAs [5, 45].

2.1.1 Connection Weight Adaptation

Learning in ANN's, also referred to as training, is accomplished by iteratively adjusting the connection weights between neurons until the desired result is achieved [5].

Training in ANNs can be categorised into supervised, reinforcement and unsupervised learning. Supervised learning makes use of training examples. The target and actual outputs are directly compared and the error of the ANN is calculated using the Mean Squared Error (MSE) function [5]. The ANN is typically provided with training examples that specify the desired output for a given set of inputs.

The ANN is considered to be trained once the error between the target and the actual outputs is sufficiently small. An algorithm based on gradient descent, such as backpropagation, is used to iteratively adjust the connection weights in order to minimise the error[5, 20].

Reinforcement learning can be a considered a special case of supervised learning, where the exact target output is not known[45]. It provides a way of programming ANNs by using reward and punishment where the ANN processes a set of inputs and is rewarded based on the accuracy of the output [22]. A simple example is placing an agent in a twsodimensional grid environment in which it needs to find the optimal set of moves to reach a destination point. The agent starts with no knowledge on how to solve the problem and moves in a random direction to get to a new state[16]. If that move decreased its distance from the destination point then the agent is rewarded and conversely punished if the distance increased. In this way, the agent associates rewards with different states and tries to maximise reward based on past experiences [29, 16].

Unsupervised learning is used when there is no information on the desired output and the learning process is based only on the correlations between the input values [45]. In cases like this, NE can be used to evolve the network weights [45].

2.1.2 Topology

The topology of an ANN is determined by the number of neurons and the interconnections between them[9]. ANN's can be classified as feed-forward or recurrent depending on the connectivity between its neurons [45]. An ANN is classified as feed-forward if data moves through the network from inputs to outputs only [16]. That is to say, if there exists a method of numbering the respective layers of the network such that all the inter-neuron connections terminate at a neuron with a greater layer number than the neuron it

started from. Conversely, an ANN is classified as recurrent if there are connections that terminate at nodes with smaller values than the source node such that it forms a feedback connection [11, 45].

2.2 Evolutionary Algorithms for adapting ANNs

EAs belong to a class of stochastic, population-based search algorithms and provide an alternative approach to conventional learning methods by drawing inspiration from Darwinian evolution by natural selection[15, 45, 16].

In conventional ANN training methods, a single solution is maintained and iteratively improved [16]. EAs, however, maintain a population of candidate solutions that allow it to sample the search space at several points simultaneously [17, 16]. This prevents the algorithm from getting stuck in a local minima and makes it suitable for complex multimodal problems[17].

Although there are several representations for encoding a chromosome, such as real values or graph encoding, this literature review is focussed on the canonical EA which uses binary encoding[11].

Parameters of the candidate ANN solutions within the population are encoded into binary strings called chromosomes [14, 33]. These are the parameters of the ANNs that are going to be evolved, such as the connection weights where each individual weight is a gene. The collection of these genes is referred to as a chromosome. Concatenating the strings in the chromosome results in the genotype representation of the ANN [11, 45, 33].

During each iteration, referred to as a generation [17], the genotype of every candidate solution is transformed to an ANN phenotype in order to evaluate its performance at a specific task using a fitness function [14].

The EA then selects the networks with the highest fitness values and applies genetic recombination operators, such as mutation and crossover, in order to produce the next generation of candidate solutions, discarding the networks with lower fitness values[38]. This process is analogous to natural selection and ensures that the most successful genotypes are propagated to the subsequent generation in the hope of these favourable traits resulting in an optimally fit individual [14]

The crossover operation is a method of performing a structured, yet random exchange of genetic material between a parent and child generation of candidate solutions. It is based on the assumption that a combination of good chromosomes from 2 fit parents will result in an even fitter individual [38]. Crossover occurs randomly according to some probability function. It selects a random point on 2 chromosomes and exchanges the collection of genes before and after that location to produce 2 offspring, each consisting of a combination of random portions of their parents' genotypes [33]. The mutation operator is used to invert binary values at some random index of a chromosome. This operation can be performed at each gene in a chromosome with some probability [33]. Mutation is used to restore genes that were previously unexplored or lost during the process of selection [38] These genetic operators correspond to the EA being able to explore and exploit the search space, where mutation allows for random exploration and cross over encourages exploitation of previous results in order to direct the search. A trade-off between these operators is needed in order maintain diversity but encourage convergence[38].

3. ENCODING SCHEMES

As mentioned in the previous section, choosing an encoding scheme is an important aspect of designing a NE system. The way in which a chromosome is encoded has a significant effect on shaping the search space, the way in which the search algorithm behaves and most importantly how the genotypes of networks are transformed into phenotypes in order to be evaluated [16].

There are several different methods of encoding genes in NE such as graph encoding and non-mating but in this literature review only the canonical binary encoding of genes is investigated according to direct and indirect encoding [11].

3.1 Direct Encoding

In this encoding scheme, all the parameters of an ANN's architecture are explicitly encoded on the chromosome as binary strings [16]. Each gene in the chromosome directly relates to a specific part of the network, allowing for a direct one-to-one mapping from the genotype to the phenotype [31, 45].

The direct encoding scheme is relatively straightforward to implement and allows for a single connection to be easily added or removed from an ANN making it suitable for precise fine-tuning of an architecture [45]. It may also accommodate rapid optimization and generation of unexplored architectures [32].

A disadvantage of using direct encoding is that it does not scale well [45]. Since the network components are mapped directly onto the chromosome there is no compression of information and would become computationally expensive with larger ANNs.

3.2 Indirect Encoding

In indirect encoding schemes, the genotype can be transformed into a phenotype using any computable function [25].

This encoding scheme functions at a higher abstraction level than direct encoding [16].

The indirect encoding scheme is also known as developmental encoding [4].

Different methods of implementing indirect encoding include using parametric functions to represent genes in a chromosome and using predetermined developmental rules to construct architectures [45], the latter usually being described by a recursive function or a production rule [25, 34].

The main advantage of this scheme is that it can represent very large networks without using large chromosomes [16].

By only encoding the necessary parts of a solution in the chromosome, the EA will focus its search on the relevant part of the solution space[10].

Indirect encoding often result in regularities wherein genetic information can be reused to influence different parts of the phenotype. Reusing genetic material aids in the scalability of a system [3].

Although indirect encoding schemes have been used in numerous applications to reduce the length of an architecture's genotype representation [24, 18], they do have several disadvantages.

Indirect encoding schemes implicitly restrict the search to the topologies that they can be expanded to since they use an indirect mapping between their genotypes and phenotypes and it is difficult to ensure that these encodings are expressive enough to include useful topologies [11].

Existing encodings such as this lack continuity in the map-

ping between genotypes and phenotypes [25]. This means that small changes in the genotype can result in large changes in the phenotype.

4. APPLICATIONS

This section will examine different applications of various NE methods in order to determine their effectiveness in solving different types of problems. These various problems include the pole balancing problem and especially tasks that require collective behaviour such as the gathering or construction tasks.

The pole balancing problem is a well known control task that has been widely used for comparing different NE methods because it is a benchmark in literature [11]. The problem is relatively straight forward, it consists of a pole hinged onto a cart with wheels. This cart is mounted on track of finite length and can only move forwards and backwards along the rails. The objective is to move the cart so as to keep the pole balanced vertically. This problem can be solved easily enough that it must be adapted in order to make it more challenging [16].

Gomez used an adapted version of the pole balancing problem by adding a second hinged pole of different length to the cart, making it a much more complex problem [16]. This was then used in an experiment to compare the performance of various NE methods.

The results obtained after performing the experiment with complete state information show that ESP and NEAT performed equally well in with respect to evaluations where SANE required 3 times as many evaluations for the same task. ESP did however have a statistically significant advantage over NEAT regarding CPU time [16].

The experiment was performed using incomplete state information. ESP and NEAT performed 3 times faster than conventional NE (CNE) methods. It was also remarked that the number of evaluations needed by CE was an order of magnitude larger than the number of evaluations needed by ESP, CNE and NEAT [16].

In another paper, Stanley remarked on similar results stating that NEAT only required a third of the number of evaluations that Q-Learning needed using complete state information. [11].

HyperNEAT has been shown to exploit intermediary regularity of a problem. This enables it to increasingly outcompete various direct encoding techniques, such as NEAT, as the regularity of the problem increases[3].

One such task is the collective gathering task which aims to replicate the efficiency and success of social insects in gathering resources [35]. The gathering task requires a team of agents to search for, collect and transport resources to some particular destination point within the environment. Gathering tasks such as this can be viewed as optimization problems that require a group of agents to optimally divide their labour on sub-tasks to derive a collective behaviour [35].

In a simple food gathering task, HyperNEAT was able to successfully evolve an agent controller that could solve the gathering task. This experiment also went on to show that these solutions were able to scale to any resolution due to their indirect encoding method [39].

HyperNEAT has also demonstrated significant success in evolving behaviours for different multi-agent tasks such as RoboCup Soccer[42] and Pursuit Evasion [4].

The collective construction task is investigated because it is a variation of the collective gathering task [43] in which HyperNEAT had produced promising results. HyperNEAT could be beneficial in the collective construction task because it requires modular and regular structures to be built [43].

During an experiment in which a simulated team of robots had to cooperatively build a structure from blocks that they gathered, HyperNEAT was able to produce effective robot team morphologies as well as evolve controllers that could be generalized to a range of different morphologies [43].

As part of an investigation into evolving quadruped gaits with HyperNEAT, Clune and Beckmann [2] compared the solutions produced by HyperNEAT and a Fixed-Topology version of NEAT (FT-NEAT). The results of this investigation showed that HyperNEAT consistently outperformed FT-NEAT. One of the reasons proposed by Clune and Beckmann [2] is that HyperNEAT is able to exploit the regularity of the robot having four legs by using the same neuron for each leg instead of trying to come up with a neuron for each leg individually [2].

4.1 NEAT

Neuroevolution of Augmenting Topologies (NEAT) is a direct encoding NE method that is used to evolve connection weights as well as topologies of ANNs [16].

NEAT is initialised with a small population of simple networks which are then gradually made more complex either by adding new connections or through mutation [13, 16].

In order to combat the problem of genomes with variable lengths, NEAT makes use of a global innovation number which is incremented whenever a new gene appears and is then assigned to that gene. This allows NEAT to keep track of the origin of each gene and thus eliminates the variable genome length problem [11]. Maintaining this historical record of each structural component in the population allows for crossover to occur between ANNs with different topologies as well as maintaining diversity between different structures by grouping networks into different species [13, 16].

Differentiating networks by species is done using a compatibility operator and gives topological innovations a chance to develop and optimise their structures before competing with members from a different species[13, 16].

4.2 SANE

Symbiotic, Adaptive Neuroevolution (SANE) is a cooperative NE method that coevolves a population of neurons instead of evolving genotypes of complete ANNs [16]. It evolves populations of neurons as well as a population of blueprints simultaneously [11]. These blueprints are used to specify how to construct networks from the neurons.

This method makes use of direct encoding and evolves network topology as well as connection weights

A network in SANE can be thought of as a team of neurons that are assembled to perform some task, thus fit neurons are those that cooperate well enough with other neurons to solve the task at hand [37]. Neurons compete based on how well the networks they are part of are performing.

An advantage of using SANE is that neurons are not bound together on a chromosome as they are in conventional NE methods. This means that a well adapted neuron won't accidentally be discarded because the network it is

part of has a poor fitness. Conversely, poorly adapted neurons won't be passed on just because their networks have a high fitness value [16].

By maintaining genetic diversity in this way, SANE prevents the premature conversion of a suboptimal solution which results in a more efficient search [37].

SANE has proven to be faster and more efficient at converging to a solution than standard NE methods in tasks that take place in domains that are continuous or have hidden state information, such as the pole balancing task as well as robot navigation tasks [16, 37].

A disadvantage to using the SANE approach is that its evaluations can result in a lot of noise. This is because it does not discriminate specialisations that are still evolving when it is building networks and selecting neurons for reproduction. This inhibits its ability to evolve recurrent networks [16].

4.3 ESP

Enforced Subpopulations (ESP) is a NE method that implements direct encoding and does not evolve topology [16].

It is an extension of SANE. It takes the idea of speciation and applies it among the neurons instead of the genotypes [37].

A key difference between ESP and SANE is that neurons in ESP only have to be evaluated according to a single role whereas the neurons in SANE are evaluated according to different roles dependent on the surrounding neurons [11].

A unique feature of ESP is that it implements burst mutation when performance has become relatively stagnant for several generations. Burst mutation saves the fittest ANN at the time of stagnation and adds noise to each of its neurons to create a new group of subpopulations in an attempt to find an optimal random modification [16].

A problem with backpropagation is deciding on the appropriate number of neurons needed for the ANN, since too few neurons will inhibit the learning process and too many neurons will result in an overfitted solution that is intolerant to noise [16]. This issue is also present in ESP since it does not evolve network topologies, but this problem is mitigated by the ability to adapt the number of neurons in the networks [16].

4.4 CE

Cellular Encoding (CE) is an indirect encoding method that evolves connections weights as well as the network topology of ANNs [16].

Each cell in CE represents a neuron in an ANN and a network is constructed by using a sequence of operations that occupy or modify these neurons [16].

This method makes use of Genetic Programming (GP) in order to evolve graph rewriting programs. These programs are used to define how ANNs are to be constructed out of cells [16]. In other words, genomes are represented by a grammar tree which is a program written in graph transformation language [11].

CE still makes use of genetic operators such as mutation and crossover to allow for an appropriate architecture to be discovered through the process of evolution, thus relieving this experimenter of the task [16].

The genetic transformations in CE are inspired by cell division in nature. Different types of connections can result in different types of cell division. Genes can be reused several times during the process of constructing a network, performing cell division at a new location each time [11].

As with most other indirect developmental encoding methods, CE has the advantage of having compact gene representation as well as being able to reuse genetic material [11].

4.5 HyperNEAT

Hypercube-based Neuroevolution of augmenting topologies (HyperNEAT) is an indirect encoding method that evolves both connection weights and topology. It uses generative encoding and results in modular, regular ANNs with greater learning capacities[43, 41].

It is used to evolve an indirect encoding of Compositional Pattern Producing Networks (CPPNs), where each CPPN is a function that produces an output based on a specific input [2]. A CPPN is used to define an ANN that represents a candidate solution [19].

The process of natural development is abstracted by using CPPNs to encode ANNs and allows for the production of complex patterns by determining the attributes of their phenotypes as functions of their geometric location [3].

The basis of CPPNs can be found in nature, where it is possible to describe patterns as compositions of functions, each function representing a stage in development [4].

HyperNEAT was introduced as one of the first NE methods to exploit geometric regularities in order to evolve ANNs on a much larger scale [39].

HyperNEAT has been demonstrated as exploiting regularity and modularity in multi-agent tasks to evolve solutions that could otherwise not have been evolved.

HyperNEAT evolved CPPNs are able to encode ANNs on a large scale by exploiting regularities that it discovers along geometric dimensions of variation [39].

It has been demonstrated that HyperNEAT is able to exploit regularity and modularity in multi-agent tasks in order to produce solutions that might otherwise not have been found [43].

It has also been shown to exploit intermediate problem regularity by producing regular ANNs which result in regular behaviour [3].

HyperNEAT is therefore potentially beneficial as structures that are to be built in collective construction tasks are modular (comprised of a set of blocks) and regular (the same sequence of blocks can be repeated) [19].

Another reason for using HyperNEAT is its success in evolving team (collective) behaviours for various multi-agent tasks including RoboCup soccer and pursuit evasion [19].

Research in generative and developmental encoding has regularly revealed that large structures can be compactly represented in DNA through the reuse of genetic material [39]. Indirect encodings such as CPPNs allow for repeated structures in the genome to be represented by a single gene that gets reused in the process of mapping the genotype to its phenotype [39]. This makes for a much more compact representation of a solution.

5. SEARCH FUNCTIONS

During the process of evolving an ANN, each individual candidate solution is evaluated and assigned a fitness value. Defining a good fitness function presents a challenge since it serves two different purposes: defining the end goal as well as guiding the EA's search through the problem space [6]. In this section, different search function approaches are

investigated, outlining their possible applications as well as their strengths and weaknesses.

5.1 Objective Search

The objective search function is perhaps the most straightforward approach. It uses the basic fitness function as an objective performance measure and aims to improve solutions by selecting the most fit individuals for mating [28].

Objective functions may direct the search towards dead ends [28]. Functions such as this are said to be deceptive and as stated by Lehman and Stanley, most goal oriented fitness functions present deceptive properties [28].

As a general rule, the more ambitious a specified task is the higher the probability that the search function will be deceived by local optima [8].

Lehman and Stanley [28] states that this is because the objective function does not reward intermediate states that could potentially lead to an optimal solution and is only concerned with the end goal[28].

5.2 Novelty Search

Where as objective search is used to achieve a static objective, novelty search aims to maintain behavioural diversity by pursuing a dynamic objective [12].

In this approach, a candidate solution is rewarded for producing behaviour different from the other individuals rather than its fitness. The behavioural diversity of a candidate solution is measured by its distance from surrounding solutions in the behaviour space [13].

This difference in behaviour is defined by a similarity function that is predetermined by the experimenter. It has been shown that novelty search can consistently find solutions faster than objective searches, particularly in deceptive search domains [26, 12].

Another advantage to novelty search is that it is able to discover a set of solutions instead of just converging to a single area in the solution space[13].

The lack of a static objective means that novelty search is addresses the problem of prematurely converging to a sub-optimal solution [13]. It has also been shown that novelty search is capable of discovering a wide range of diverse solutions with less complex ANNs than the solutions produce by objective search [13].

The high level of complexity inherent in real life problems tends to produce deceptive fitness landscapes with multiple local optima and novelty search has been shown to be immune to deception by preventing the search from getting stuck in a local optimum [13].

Using novelty search may result in a lot of time being wasted searching novel regions that may not contribute to an optimal solution [13].

5.3 Hybrid Search

Hybrid search, as the name suggests, is a search method which combines EAs as well as local search [1].

By combining these two different types of searches, a hybrid algorithm is able to leverage beneficial capabilities of each algorithm. The EA provides the ability of being able to find a candidate solution that is near a global optimum and the local search provides the functionality of fine tuning a solution produced by the EA to reach the nearest local minimum [1].

A hybrid variant of novelty search, called minimal criteria novelty search (MCNS), in which an individual would have to meet some or other minimal criteria dependent on their domain in order to be selected for reproduction [27], was tested in two maze navigation tasks and would consistently outperform both the objective search and the novelty search [27].

This specific hybrid method does have its drawbacks. First off, choosing the minimal criteria within a particular domain space is difficult and requires consideration. Secondly, if no candidate solutions meet the minimum criteria, the search is effectively random and lastly, if the criteria are too strict then it may be difficult to mutate suitable individuals within the constraints resulting in wasted evaluations [13].

6. CONCLUSIONS

By looking at the information presented in this literature review it is possible to conclude that further investigation into the applications of HyperNEAT could be beneficial to the development of multi-robot teams to solve a collective task.

HyperNEAT has been shown to be successful in evolving solution behaviours for various multi-agent tasks [42, 4], its indirect encoding method means that it can reuse genes and therefore have a more compact representation of the chromosomes and the genomes [11, 39].

Not only that, but HyperNEAT has also been used to produce robust controllers that successfully solved a collective construction problem and could be scaled to any resolution as well as being easy to generalise [43].

There are numerous real world applications for multi-robot teams that perform a task that requires collective behaviour and the success of HyperNEAT in problems similar to these scenarios make it an ideal candidate for further research into evolving the controllers for these robots.

All these desirable properties of HyperNEAT, along with its proven ability to exploit regularity of a problem [2], form the basis for a further investigation.

Subsequent investigation should be conducted in order to determine the effect of using objective, novelty and hybrid searches with HyperNEAT when evolving controllers for multi-robot teams to solve a collective construction task. This is part of a larger team project in which each of the members are to investigate different NE methods to evolve controllers for multi-robot teams with different sensory configurations.

7. REFERENCES

[1] P. A. Castillo, J. J. Merelo, M. G. Arenas, and G. Romero. Comparing evolutionary hybrid systems for design and optimization of multilayer perceptron structure along training parameters. *Information Sciences*, 177(14):2884–2905, 2007.

[2] J. Clune, B. E. Beckmann, C. Ofria, and R. T. Pennock. Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 2764–2771. IEEE, 2009.

[3] J. Clune, K. O. Stanley, R. T. Pennock, and C. Ofria. On the performance of indirect encoding across the continuum of regularity. *Evolutionary Computation, IEEE Transactions on*, 15(3):346–367, 2011.

[4] D. B. D'Ambrosio and K. O. Stanley. Generative encoding for multiagent learning. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 819–826. ACM, ACM, 2008.

[5] J. E. Dayhoff and J. M. DeLeo. Artificial neural networks. *Cancer*, 91(S8):1615–1635, 2001.

[6] S. Doncieux and J.-B. Mouret. Beyond black-box optimization: a review of selective pressures for evolutionary robotics. *Evolutionary Intelligence*, 7(2):71–93, 2014.

[7] A. E. Eiben and J. E. Smith. Introduction to evolutionary computing. 53, 2003.

[8] S. G. Ficici and J. B. Pollack. Challenges in coevolutionary learning: Arms-race dynamics, open-endedness, and mediocre stable states. In *Proceedings of the sixth international conference on Artificial life*, pages 238–247, 1998.

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

[10] M. Gen, R. Cheng, and S. S. Oren. Network design techniques using adapted genetic algorithms. *Advances in Engineering Software*, 32(9):731–744, 2001.

[11] J. Ghosh, B. J. Kuipers, and R. J. Mooney. Efficient evolution of neural networks through complexification. 2004.

[12] J. Gomes, P. Mariano, and A. L. Christensen. Devising effective novelty search algorithms: A comprehensive empirical study. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, pages 943–950. ACM, 2015.

[13] J. Gomes, P. Urbano, and A. L. Christensen. Evolution of swarm robotics systems with novelty search. *Swarm Intelligence*, 7(2-3):115–144, 2013.

[14] F. J. Gomez, D. Burger, and R. Miikkulainen. A neuro-evolution method for dynamic resource allocation on a chip multiprocessor. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, volume 4, pages 2355–2360. IEEE, 2001.

[15] F. J. Gomez and R. Miikkulainen. Solving non-Markovian control tasks with neuroevolution. In *IJCAI*, volume 99, pages 1356–1361, 1999.

[16] F. J. Gomez and R. Miikkulainen. Robust non-linear control through neuroevolution. 2003.

[17] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, 16(1):122–128, 1986.

[18] S. A. Harp, T. Samad, and A. Guha. Towards the genetic synthesis of neural network. In *Proceedings of the third international conference on Genetic algorithms*, pages 360–369. Morgan Kaufmann Publishers Inc., 1989.

[19] M. Hausknecht, P. Khandelwal, R. Miikkulainen, and P. Stone. HyperNEAT-GGP: A HyperNEAT-based Atari general game player. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 217–224. ACM, 2012.

[20] G. E. Hinton. Connectionist learning procedures. *Artificial intelligence*, 40(1):185–234, 1989.

[21] C. Igel. Neuroevolution for reinforcement learning using evolution strategies. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 4, pages 2588–2595. IEEE, 2003.

[22] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, pages 237–285, 1996.

[23] B. Khoshnevis and G. Bekey. Automated Construction Using Contour Crafting–Applications on Earth and Beyond. *Nist Special Publication Sp*, pages 489–494, 2003.

[24] H. Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems Journal*, 4:461–476, 1990.

[25] J. Koutnik, F. Gomez, and J. Schmidhuber. Evolving neural networks in compressed weight space. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 619–626. ACM, 2010.

[26] J. Lehman and K. O. Stanley. Exploiting Open-Endedness to Solve Problems Through the Search for Novelty. In *ALIFE*, pages 329–336, 2008.

[27] J. Lehman and K. O. Stanley. Revising the Evolutionary Computation Abstraction: Minimal Criteria Novelty Search. *Proc. of the International Conference on Genetic and Evolutionary Computation (GECCO'10)*, (Gecco):103–110, 2010.

[28] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[29] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the eleventh international conference on machine learning*, volume 157, pages 157–163, 1994.

[30] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[31] R. Miikkulainen. Evolving Neural Networks. In *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '10, pages 2441–2460, New York, NY, USA, 2010. ACM.

[32] G. F. Miller, P. M. Todd, and S. U. Hegde. Designing neural networks using genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 379–384. Morgan Kaufmann Publishers Inc., 1989.

[33] M. Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[34] E. Mjolsness, D. H. Sharp, and B. K. Alpert. Scaling, machine learning, and genetic neural nets. *Advances in applied mathematics*, 10(2):137–163, 1989.

[35] G. S. Nitschke, M. C. Schut, and A. E. Eiben. Emergent specialization in biologically inspired collective behavior systems. In *Intelligent complex adaptive systems*, pages 100–140. 2007.

[36] G. S. Nitschke, M. C. Schut, and A. E. Eiben. Evolving behavioral specialization in robot teams to solve a collective construction task. *Swarm and Evolutionary Computation*, 2:25–38, 2012.

[37] D. Polani and R. Miikkulainen. Fast reinforcement learning through eugenic neuro-evolution. *University*

of Texas at Austin, Austin, TX, 1999.

[38] M. Srinivas and L. M. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, 24(4):656–667, 1994.

[39] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

[40] K. O. Stanley and R. Miikkulainen. *A taxonomy for artificial embryogeny.*, volume 9. 2003.

[41] P. Tonelli and J.-B. Mouret. On the relationships between generative encodings, regularity, and learning abilities when evolving plastic artificial neural networks. *PloS one*, 8(11):e79138, 2013.

[42] P. Verbancsics and K. O. Stanley. Evolving Static Representations for Task Transfer, 2010.

[43] J. Watson and G. Nitschke. Evolving Robust Robot Team Morphologies for Collective Construction. In *Computational Intelligence, 2015 IEEE Symposium Series on*, pages 1039–1046. IEEE, 2015.

[44] J. Werfel, Y. Bar-Yam, D. Rus, and R. Nagpal. Distributed construction by mobile robots with enhanced building blocks. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2787–2794. IEEE, 2006.

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

[46] B. Yegnanarayana. *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.

[47] G. Zhang, B. E. Patuwo, and M. Y. Hu. Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting*, 14(1):35–62, 1998.