Investigating the ability of objective, novelty and hybrid fitness functions to evolve multi-robot team controllers to solve complex collective behaviour tasks

A. Ruben Putter
University of Cape Town
Rondebosch 7701
Cape Town, South Africa
Supervisor: Dr. Geoff Nitschke
PTTAND010@myuct.ac.za

ABSTRACT

This paper investigates the ability of objective, novelty and hybrid fitness functions to evolve multi-robot team controllers that are able to solve a collective construction task using the HyperNEAT evolutionary algorithm. These approaches are tested using tasks of increasing complexity and that require different levels of cooperation within a robot team. It was found that, although they produced relatively similar task performance results for the easier tasks, the controllers that were evolved with novelty fitness outperformed the other approaches in the most difficult task, further establishing its ability to deal with deceptive problem domains.

Keywords

HyperNEAT; Neuroevolution; Artificial Neural Networks; Evolutionary Algorithms; Evolutionary Robotics; MultiRobot Team; Collective Construction Task;

1. INTRODUCTION

One of the central themes of research in mathematics and computer science is the design and development of automated problem solvers [8].

The current trend in increasing levels of computerisation coupled with the increasing complexity of tasks that need to be performed has led to a growing demand for the automated production of these problem solvers [8].

Given the level of complexity and non-linearity of real world problems, traditional controller design is impractical since they rely on linear mathematical models [15].

While it may be possible to solve many complex tasks using a single sophisticated robot, there are several advantages to using a multi-robot team instead. Some of these advantages include being able to use much simpler designed robots in the team instead of one very expensive robot as well as having increased fault tolerance since there is no longer just one point of failure (each robot in the team is a point of failure) [2].

The complex nature of the intricate dynamics inherent in producing self-organised behaviour makes it practically impossible to design these multi-robot team controllers by hand [13].

The field of evolutionary robotics aims to automate the production of robot controllers that are capable of producing complex behaviours [27].

Neuroevolution (NE) is a method of artificially evolving Artificial Neural Networks (ANNs) by implementing Evolutionary Algorithms (EAs) to perform tasks such as evolving the network connection weights as well as the network topology[11, 38].

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 [28].

The different NE methods can be roughly divided into two categories with respect to the encoding used to map between an individual's genotype and phenotype, namely direct and indirect encoding.

This experiment only focusses on the indirect (also called generative or developmental) encoding method called HyperNEAT [30].

HyperNEAT was chosen for these experiments due to its proven success in evolving team (collective) behaviours for various cooperative multi-agent tasks including RoboCup soccer and pursuit evasion [16].

It was also chosen because it has been shown to outperform the different direct encoding methods in a variety of different tasks [34].

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 [9]. 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 [22]. The second is called novelty search which 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 novelty with an objective 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 [28].

This task was chosen as a case study because of its inherent relative complexity compared to other existing collective behaviour tasks (such as the collective gathering task) [28].

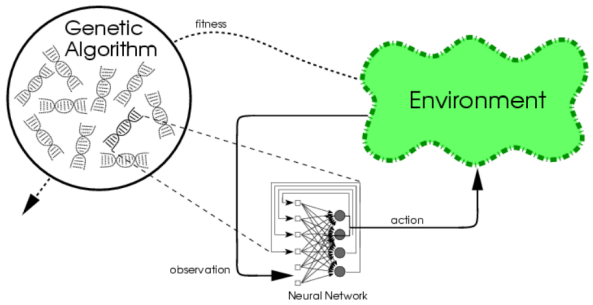

Figure 1: A diagram showing the generate and test implementation of NE

It was also chosen because of its many real world applications such as producing low-cost housing and reducing accident rates due to human error [20]. It can also be applied to using robot teams to terraform other planets and to perform construction in areas that are dangerous or inaccessible to humans [37].

1.1 Research Objectives

This experiment is aimed at investigating the impact that different fitness function approaches have on directing HyperNEAT evolution of multi-robot ANN controllers to solve increasingly complex collective construction tasks.

2. RELATED WORK

2.1 Neuroevolution

Neuroevolution (NE) is an evolutionary approach to modifying the weights and topologies of a neural network in order for it to learn how to perform a specific task [25].

This is accomplished by way of applying Evolutionary Algorithms (EAs) to evolve the inter-neuron connection weights and topologies of Artificial Neural Networks (ANNs) [11].

ANNs are a simplified model of how a biological brain functions[24]. 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 [15].

Instead of adapting a single controller to try learn a behaviour, NE maintains a population of individuals to search the space of possible successful controllers [15].

By evolving a population of ANNs, NE searches through the space of possible controllers by decoding each genotype into its phenotype (the ANN controller) which is then evaluated using a fitness function and selecting the most successful individuals for reproduction, a process analogous to evolution through natural selection [15, 14].

A description of the general generate-and-test loop that is used by most NE method is outlined below [25]:

1. Generate the initial population G(0) at random and set $i = 0$;
2. REPEAT
   (a) Evaluate each individual in the population;
   (b) Select parents from *G(i)* based on their fitness
   (c) Apply genetic operators to parents and produce offspring which form *G(i+1)*, the subsequent generation.
   (d) $i = i + 1$ -> advance the time step and repeat
3. UNTIL the termination criteria has been reached (a fitness threshold or specific number of generations)

In order to evaluate a population, each genotype is respectively decoded into its phenotype (ANN) which is then employed in the required task and its performance over time is measured and used to calculate a fitness function [25].

This process constitutes an intelligent parallel search that results in increasingly fitter genotypes [25].

NE is a policy search method for Reinforcement Learning (RL) type problems where the domain is continuous and the state is only partially observable [25].

Compared to other ANN learning methods, such as Reinforcement Learning [33] and Backpropagation [19], NE is highly general which means ANNs are able to learn a behaviour without requiring explicit behavioural targets [25].

The primary motivation for using NE is to overcome/compensate for the disadvantages of other ANN learning approaches such as being able to train ANNs to make sequential decision tasks with sparse reinforcement information [25]. NE is also not as susceptible to getting stuck in local minima in the same way that gradient-descent algorithms are [15].

The different NE methods can be categorised according to the way in which the phenotypes are encoded into their respective genotypes. The categories are split into two, namely for direct and indirect encoding (also referred to as a generative or developmental encoding) [10].

The direct encoding scheme is characterised by having all the ANN parameters explicitly represented on the encoded chromosome (or genome), these parameters can be mapped directly to the phenotype [15].

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, 32].

2.2 HyperNEAT

HyperNEAT (Hypercube-based Neuroevolution of Augmenting Topologies) [4], is an evolutionary algorithm that uses a novel indirect encoding called Compositional Pattern Producing Networks (CPPNs), where each CPPN is a function that produces an output based on a specific set of inputs, to evolve ANN controllers [7].

An evolutionary algorithm (EA) is a population-based stochastic search method based on the principles of Darwinian evolution through natural selection [10].

Instead of being directly applied to ANNs, HyperNEAT works by using the NEAT algorithm (Neuroevolution of Augmenting Topologies [31]) to evolve the CPPN, which is a genotype (or network) representing the connections of a possibly much larger network [35]. In other words, a CPPN is used to define an ANN that represents a candidate solution in the population [17].

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].

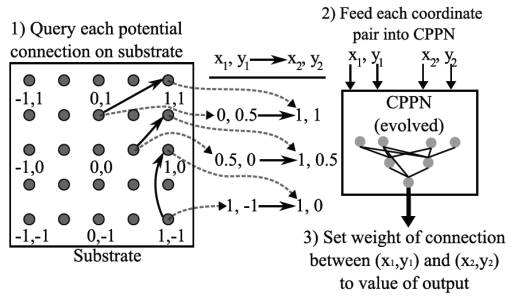

Figure 2: Every potential connection in the substrate is queried in order to determine its location and associated weight

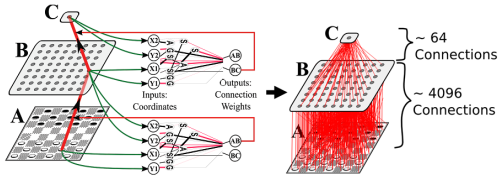

Figure 3: Example substrates for HyperNEAT when learning to play checkers

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 [5].

The connectivity patterns that are produced by a CPPN is called a substrate where each queried point is a neuron in an ANN [29]. The process of querying each point in the substrate to produce the ANNs is shown in figure 2.

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 [36].

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) [16].

Another reason for using HyperNEAT is its success in evolving team (collective) behaviours for various multi-agent tasks including RoboCup soccer and pursuit evasion [16].

2.3 Fitness 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].

This paper is aimed at investigating three different types of fitness functions, namely objective, novelty and hybrid fitness.

The objective search 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 [22]. Objective functions have the disadvantage of becoming trapped in local optima, which would cause the EA's search to stagnate and converge to a suboptimal solution [21].

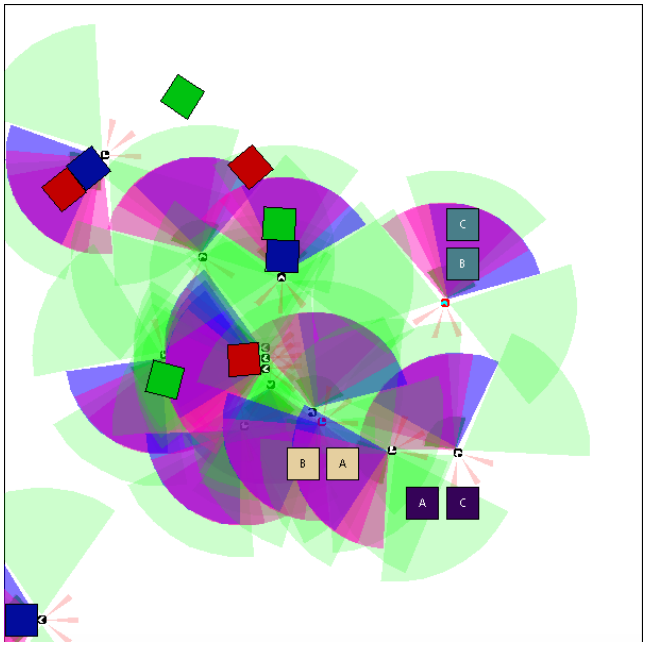

Figure 4: A screenshot of the simulated environment showing the robots along with the resources, both connected to construction zones and free-standing. The coloured arcs indicate the range and field of view of the different sensors that are attached to the robots. The blocks with a type label are connected in construction zones. It is also possible to see three individual robots cooperating in order to try and move the C-type block,

Novelty search is a technique of guiding the evolution towards increasing behavioural novelty in a population by rewarding individuals based on how different their behaviour is compared to the other individuals in its generation [12, 22]. Novelty search maintains behavioural diversity by pursuing a dynamic objective [12].

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.

3. METHODS

This section of the paper is aimed at explaining the various methods used in order to set up the experiments.

3.1 Simulated Environment

The MASON Multiagent Simulation Toolkit was used to implement the simulated environment [23]. This toolkit is implemented in Java and produces results that are consistent across various platforms [23] which is important since these experiments were programmed and run on computers using different operating systems. The toolkit also uses self-contained models that are independent of their visualisations. One of the main reasons for using the MASON library is that it can be run inside other Java frameworks [23], making it ideal to work with the Encog Machine Learn-

ing Framework which is discussed later on in this section.

The simulated environment provides a 20x20 metres continuous 2-dimensional area within which the entities can interact.

A screenshot of the simulated environment is shown in figure 4.

The MASON library also provides a GUI which allows the experimenter to start, pause, stop, reset and step through the simulation. This GUI does not get used during the experiments as it would have been too computationally expensive, but is used for observing the behaviour of the evolved controllers as well as capturing screen-shots of the environment during the evaluation phase.

The physics of the simulated world were implemented using the JBox2D library for Java [26].

It should also be noted that the continuous environment is also sub-divided into an underlying discretized grid representation. This grid representation is used for accurately positioning resources in the construction zones and efficiently comparing the structures built by the various controllers. How the discrete grid gets used for these functions is described in more depth in the subsequent sections.

3.1.1 Construction Blocks

For these experiments, three different types of construction blocks were used. These blocks (also referred to as "resources") are shown in figure 5. These detection points are used to determine whether or not two construction blocks are positioned in such a way that they can be connected to each other.

In order for two resources to be able to connect to each other, they must satisfy two criteria, namely: a) the blocks must be properly aligned with respect to each other's orientation (in other words, their neighbouring edges must be parallel to each other) and b) the distance between the blocks must be less than a predetermined threshold.

Once these criteria are met, the two construction blocks are snapped together and fixed in position. When blocks are connected, they are "snapped" into their nearest discretized location so that each block occupies a cell in the underlying grid representation.

Figure 4 shows an example of the different construction blocks scattered throughout the simulated environment.

The different types of construction blocks require different numbers of robots to move them. For example, blocks of type A can be picked up and moved by only one robot whereas blocks of type B and C require two and three robots respectively to move them.

3.1.2 Connection Schemas

The different types of construction blocks can only be connected to specific types of resources according to a predefined connection schema. The connection schema specifies the number of each type of construction block that is to be placed in the environment for a specific experiment run.

It also indicates which type of block can be connected to each respective side of a block, that is to say, for each construction block type placed in the environment, the connection schema indicates which block type can be attached to each of its respective sides.

As can be seen in figure 5, the permissible connections are denoted by a single letter indicating block type. When block types have no letters on the sides it means that there were

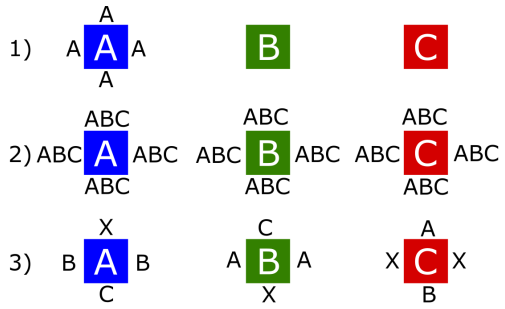

Figure 5: Construction schemas used in the experiments as well as showing the different types of blocks

Table 1: Schema parameters for the easy, medium and difficult schema where ψ is the amount of cooperation required, γ is the complexity of the construction schema and Φ is the overall schema complexity. The calculations for these values are shown in the text

Schema number	complexity	number of A/B/C	ψ	γ	Φ
1	easy	15 / 0 / 0	0	0	0
2	medium	5 / 5 / 5	1	0	0.5
3	difficult	5 / 5 / 5	1	0.789	0.89

no blocks of that type in the environment (as with block types B and C in the first schema). When a block type has an 'X' on a side, it means that no connections of any kind are permitted on that side (as in the third schema).

The complexity of a given schema can be formalised using the following equation:

$$\psi = \begin{cases} 0 & \text{if only A resources are present} \\ \frac{1}{2} & \text{if B resources are present} \\ 1 & \text{if C resources are present} \end{cases} \quad (1)$$

$$\gamma = \left(1 - \frac{\bar{a}}{\eta}\right)\left(\frac{A}{N}\right) + \left(1 - \frac{\bar{b}}{\eta}\right)\left(\frac{B}{N}\right) + \left(1 - \frac{\bar{c}}{\eta}\right)\left(\frac{C}{N}\right), \quad (2)$$

$$\Phi = \frac{1}{2}\psi + \frac{1}{2}\gamma. \quad (3)$$

where $\bar{a}$, $\bar{b}$ and $\bar{c}$ refer to the number of sides that blocks of type A, B and C appear in the construction schema for a block respectively.

η refers to the number of different block types present in the environment multiplied by four (the number of possible connection sides in a schema).

A, B and C simply refer to the number of blocks of that type present in the schema and N is the total number of resources in the environment which is kept constant in each schema and shown in the table 2.

The first schema consists of only type A construction blocks. It also shows that a block of type A can be connected on any side to another type A block. This makes it the easiest schema since there are no underlying connection rules to be learnt and no cooperation is required to move any of the type A blocks.

The second schema contains all of the different types of blocks and indicates that each block type can connect to any

other type on any of its sides. While there are no underlying connection rules to be learnt, the second schema is still more complex than the first one since it requires that robots learn to cooperate in order to move the different block types.

The third and final schema is the most challenging. It makes use of each type of block and imposes a complex set of connection rules that are different for each block type. This means a controller will need to develop cooperative behaviour amongst its team of agents as well as learn the underlying connection rules in order to successfully solve the task.

3.1.3 Collective Construction Task

The collective construction task requires that a team of agents cooperate by coordinating their behaviours in order to assemble various structures within their environment [28].

In this case, a team of homogeneous robots needs to search for various types of construction blocks scattered throughout the environment and connect them in order to construct a single large structure.

The collective construction task can be sub-divided into three smaller sub-tasks:

First, the robots need to search the environment in order to find the loose construction blocks.

Second, once a robot has located a block, it needs to pick it up and move it towards a construction zone. If there is no previous construction zone, then it needs to move the block towards another block in order to start a new construction zone.

Third, the robot must then try to connect the block it's carrying either to an existing construction zone or another block. Once the robot has successfully connected the block to a construction zone, it releases the block and searches for another free-standing construction block.

The robots need to cooperate in order to collect and construct the heavier block types.

The controller also needs to learn the underlying connection rules specified by the schema in order to be able to connect the appropriate types of blocks to each other.

When two construction blocks are connected to each other, they create a new construction zone.

A construction zone refers to an already existing structure that consists of two or more connected resources. Three individual construction zones are shown in figure 4 and are indicated by the connected blocks changing to the same colour as well as having a label indicating the block's type.

Resources that are part of a construction zone get fixed in place and can not be moved or disconnected.

A new construction zone gets created when a pair of blocks (neither of which can be part of an existing construction zone) are connected. This only gets done for the first three pairs of blocks in order to encourage the development of a singular large structure instead of several smaller ones.

This means that the location of construction zones is different with each simulation run and depends on where the robots connect the first two construction blocks.

The complexity of the collective construction task is controlled by the different connection schemas. By requiring the robots to cooperate in order to move certain block types and implementing connection rules, it is necessary to evolve more complex controllers that account for cooperative interactions which is non-trivial for most EAs.

The controllers that were produced using HyperNEAT and the various fitness functions were evaluated based on how successful they were at solving the collective construction task. A controller's task performance is defined by the number of different types of blocks that it managed to connect to a construction zone.

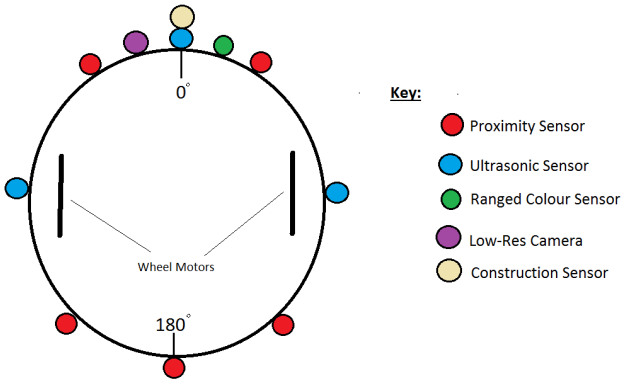

Figure 6: A diagram showing which sensors were used in the morphology as well as their relative positions on the robot.

The most successful controller produced by each approach out of the ten experiment runs is then found for each construction schema.

Each of these controllers is then tested in the simulator for a set number of timesteps indicated in table 2. At the end of a simulation run, the number of construction blocks that are successfully connected is recorded and saved to a file. This process is repeated twenty times for each controller.

3.1.4 Robots

The morphology that was used can be seen in the diagram 6. It shows the different types of sensors that were used in the configuration as well as their relative positions on the robot.

The robots that were used in this simulation were abstracted and simplified versions of the Khepera robot.

Each robot is equipped with a set of sensors that make up its morphology.

Each of the robots also has two wheel motors which are used to control its movement direction and speed. This gives the robot the ability to rotate and move in any direction on the two-dimensional environment.

3.2 Controller Adaptation: HyperNEAT

In order to conduct this experiment, the homogeneous robot team controllers were evolved using the HyperNEAT evolutionary algorithm implemented with objective, novelty and hybrid search.

3.2.1 HyperNEAT

For this experiment, the HyperNEAT evolutionary algorithm was implemented using the Encog Machine Learning framework for Java [18].

The Encog framework was used to perform all the evolutionary processes. This includes selecting the parent individuals from the current generation, performing mutation and crossover and producing the new generation of individuals.

The framework also makes use of multi-threading techniques so that controller evolution can be optimised on ma-

chines with multi-core processors.

At the end of each evolution phase, the Encog framework produces a genome that is then decoded into an Artificial Neural Network (ANN). This ANN is then evaluated in the MASON simulator and a fitness score is calculated and associated with the current network.

The Encog framework requires that the experimenter provide a function to evaluate the performance of a controller in order to calculate a score. This function simply needs to return a fitness value, making it very simple to integrate into the framework. This makes Encog ideal for testing different fitness functions since they can be easily interchanged.

3.2.2 Objective Search

A controller's reward is calculated based on the number of each type of block that it was able to successfully connect to a construction zone.

The objective fitness function can be formalised as follows:

$$f(x) = \sum_{i=1}^{3} b_i \times C_i \tag{4}$$

where f is the total objective score, i iterates over the different types of construction blocks, b_i refers to the number of blocks of type i that were successfully connected to a construction zone and C_i is the associated weight for block type i.

The specific weights used for the various block types are specified in table 2.

This raw objective score is the normalised with respect to the ideal score for that schema in order to get a final fitness value of [0,1].

This ideal score is calculated using exactly the same weighted sum as in equation 4 except b_i refers to the total number of blocks of type i that are available in the environment.

3.2.3 Novelty Search

With novelty search, the fitness of an individual can only be calculated relative to other individuals. This is because novelty search is used to measure how different the behaviour of an individual is compared to the behaviour of others.

The behaviour of an individual therefore needs to be quantified using some sort of metric. In order to implement novelty search, it is necessary to first come up with an appropriate similarity measure to be able to compare resultant behaviour of controllers [12].

The behavioural metrics that were used in this experiment are as follows:

- structure of the resultant construction zone
- the order in which the different block types were connected
- the endpoints of the individual robots

The resultant construction zones produced by the various controllers were compared by using the underlying discretised grid to represent the structures as a two-dimensional array. The differences between the structures were found by iterating over these arrays and checking if the respective cells contained the same type of construction block. It should be noted that an empty cell (no block was connected in that location) is treated as a block type as well. A counter is incremented for each difference that is found.

This metric was used in order to encourage robots to try and connect the different types of blocks in as many different ways as the current schema would allow.

Whenever a block was connected to a construction zone, its type would be recorded and added to an ordered list. This construction order was compared by iterating over this list and checking if each element in the respective lists are of the same type. A counter is incremented each time a difference is found.

At the end of each simulation run, the coordinates of each robot in the team is recorded. The ending locations of the i^{th} robot for each team is then compared by calculating the square Euclidean distance between them. These distances are then summed for each time and divided by the number of robots in a team.

This metric was implemented in order to encourage the robots to explore their environment.

In order to calculate the novelty score of an individual's phenotype, it gets compared to every other individual's phenotype in its generation as well as the phenotypes maintained in the novelty archive.

The novelty archive is a collection of the most novel phenotypes found in previous generations. At the end of each generation, the phenotype with the highest novelty score was added to the archive.

This collection was used to increase the required behavioural diversity.

The novelty score of an individual is determined by how similar it is to the other individuals in the generation and the archive. By using the behavioural similarity measure, an individual's score is calculated by finding the average distance between it and its k-nearest neighbours in the behaviour space [12]. This calculation is formalised below:

$$\rho(x) = 1/k \sum_{i=1}^{k} noveltyDist(x, \mu_i) \tag{5}$$

where ρ represents the final novelty score for an individual x, k represents the number of nearest neighbours that need to be evaluated. This value is specified in table 2. Finally, μ_i represents the i^{th} nearest neighbour in the novelty space. The $noveltyDist$ is the novelty score that was calculated using the aforementioned behavioural metrics.

3.2.4 Hybrid Search

Hybrid search was implemented as an equally weighted average between the objective function and the novelty function [12].

Whenever an individual controller was being evaluated, an objective score as well as a novelty score is calculated based on its resultant phenotype behaviour. This values are then added together and averaged.

3.3 Experimental Design

An experiment refers to the application of HyperNEAT to evolve a population of 150 ANN controllers over 100 generations, where each generation runs five task trials in the simulator in order to evaluate an individual.

A single experiment run essentially consists of two main phases, the Evolutionary Phase and the Evaluation Phase, that get repeated for 100 generations.

A generation is completed when all of the ANN's in the population have been evaluated.

Table 2: EA and Simulation Parameters

Simulation runs / Generations per run	10 / 100
Epochs (team lifetimes) per generation / Iterations per epoch	5 / 1000
Genotype population size / elite portion / Robot team size	150 / 0.3 / 15
ANN connection weight range	$[-5.0, 5.0]$
Robot size (diameter) / Gripping distance**	0.015 / 0.002
Wait for help (cooperation) time	350 (simulation time steps)
Initial robot / resource positions	Random
Environment width x height / Type / Max number of CZs	1.0 x 1.0 / Continuous / 3
Construction area dimensions (row x col) / Type / padding	20 x 20 (0.05 x 0.05 per cell) / Discrete grid / 0.0125
A / B / C resource size (Width / Height)	0.05 x 0.05 (for all three types)
Robots required to push type A / B / C resources	1 / 2 / 3
Infrared sensor range	(0.0, 1.0]
Ultrasonic sensor range	[0.2, 4.0]
Infrared sensor Field Of View (FOV)	0.2 Radians
Ultrasonic sensor FOV	1.22 Radians
Connection weight mutation probability / mutation range	0.493/ $[-5.0, 5.0]$
Initial / new node connection density	0.5 / 0.1
Crossover probability	0.5
Sensory input nodes / Motor output nodes	11 / 2
Total number of resources	15
Weights for block A/B/C	0.3/0.6/1
K-nearest neighbours	10
Add node mutation probability	0.25
Add/remove link mutation probability	0.8/0.002
Timesteps for evaluation simulation run	15 000

During the Evolutionary Phase, the Encog framework is used to evolve a generation of controllers using HyperNEAT and the various crossover and mutation operators, as shown in 2, in order to produce a successive generation of controllers.

A generation is completed once all of the ANN controllers in the population have been evaluated.

During the Evaluation Phase, each individual in the new generation of controllers is tested by running five task trials in the simulator. Each individual controller is implemented in a homogeneous team of robots and then tested in the simulated environment for a specific number of timesteps, which are indicated in table 2. At the end of a simulation run, the scoring function is used to calculate a fitness for the controller.

For the Objective function, the scores for individual trials are summed and averaged over the number of trials in order to get the final fitness value for that controller.

For the Novelty fitness function, only the highest novelty score produced in the trials is recorded.

And finally, for the Hybrid fitness function, both the average objective score and the most novel score are used to calculate the final hybrid score of a controller.

This process is repeated ten times for each of the three connection schemas which are shown in figure 5.

This means that the experiment gets performed a total of thirty times for each fitness function.

The robot team was operated by a homogeneous controller which means that each robot in a team uses its own instance of the same evolved network. In other words, there is only one ANN controller per robot team.

Each robot team consisted of fifteen individual robots.

The table 2 shows the various EA and simulation parameters that were used to conduct this experiment. All of the distances and measurements have been normalised so as to be represented as a fraction of the environment dimensions.

All of the robots used the same morphology (sensor configuration) throughout all of the experiments. This morphology is shown in figure 6.

4. RESULTS AND DISCUSSION

This section presents the experimental results of running the various best performing network controllers for a test period in the simulator.

Various statistical tests were applied to the data so as to compare the differences in task performance for the robot teams evolved using the different fitness functions as well as to determine the effect that schema complexity has on the task performance.

In order to determine whether or not the data sets conformed to a normal distribution, the Kolmogorov-Smirnoff test was applied. The KS-Test showed that not all the produced data sets are normally distributed. Consequently, this means that the Mann-Whitney U Test must be used in order to determine if there is a statistical difference between the data sets.

The Mann-Whitney U Test was applied to the following pairs of data sets to compare the different approaches for the first schema:

- Task performance of hybrid search and novelty search
- Task performance of hybrid search and objective search
- Task performance of objective search and novelty search

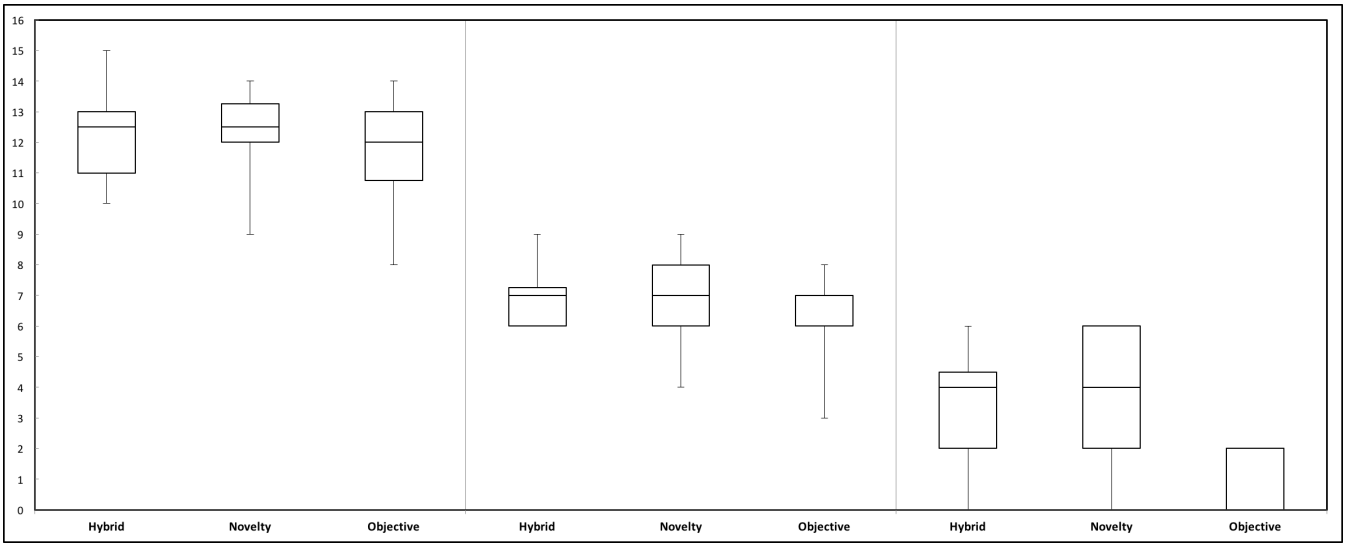

Figure 7: A box plot showing the results of the twenty evaluation runs for each approach and for each construction schema

Table 3: Table showing whether or not there is a statistically significant difference in the task performance for the approaches. A ✓ represents a statistically significant difference

	Objective	Novelty	Hybrid
Objective	X	X	X
Novelty	X	X	X
Hybrid	X	X	X

Table 4: Table showing whether or not there is a statistically significant difference in the task performance for the approaches for schema 3. A ✓ represents a statistically significant difference

	Objective	Novelty	Hybrid
Objective	X	✓	✓
Novelty	✓	X	X
Hybrid	✓	X	X

This was repeated for each of the second and third schema as well.

In order to determine the effect of task complexity on the various fitness functions, the following data sets are compared for statistical differences:

- Task performance of hybrid search using schema 1 and hybrid search using schema 2
- Task performance of hybrid search using schema 1 and hybrid search using schema 3
- Task performance of hybrid search using schema 2 and hybrid search using schema 3

This was repeated for objective and novelty search.

The task performance results produced during the evaluation runs are shown in figure 7. It shows the maximum and minimum number (as well as median and respective quartiles) of construction blocks that were successfully connected by each approach for every construction schema.

The plot is divided into three sections in order to separately show the results for the respective schemas.

4.1 Comparison of Search Mechanisms

The results for these Mann-Whitney tests are shown in the tables 3 and 4 , indicating whether or not there was a significant statistical difference between the different data sets.

The Mann-Whitney tests for the first and second schema produced the same results and are therefore represented in the single table 3.

There was a significant statistical difference between the performance of hybrid search and objective search for the third schema. By looking at the data shown in the box plot it is easy to see that the hybrid search outperforms objective search in the more complex problem domain.

There was also a statistical difference between the performance of novelty search and objective search for the third schema. This indicates that objective search becomes significantly less successful in increasingly complex tasks and that the search for novel behaviour is effective for searching through deceptive problem domains.

This is further supported by the fact that there is also a significant statistical difference between hybrid and objective search for the third schema, indicating that even only partially incorporating novelty search into a fitness function provides an advantage over the traditional objective reward functions.

This can also be observed by looking at the task performances shown in figure 7. It is easy to see that for the first two schemas each of the approaches produce relatively similar performing controllers, but there is a very discernible decrease in task performance for all the approaches, with objective search dropping the most, almost not connecting any blocks at all.

Table 5: Table showing whether or not there is a statistically significant difference in the task performance for objective, novelty and hybrid search across the schemas used. A ✓ represents a statistically significant difference

	Schema 1	Schema 2	Schema 3
Schema 1	X	✓	✓
Schema 2	✓	X	✓
Schema 3	✓	✓	X

4.2 The Effect of Schema Complexity

The Mann-Whitney U Test was applied to the data sets for the different approaches across the various schemas to determine if there was a significant statistical difference between an approach's task performance using schemas of varying complexity. The results for these tests are shown in table 5.

The results of the Mann-Whitney U Test was the same for all the approaches and so are shown in a single table 5.

The fact that there was a statistically significant difference for all of the approaches shows that the increasing complexity of the underlying connection rules and the levels of cooperation required by the different block types makes it increasingly difficult for the EA to find a successful controller.

This is supported by the information shown in figure 7 as it is easy to see how the overall task performance decreases for the various schemas.

It should be noted that the difference in task performance for novelty search between the second and third schemas is significantly less than that of the other approaches, indicating that novelty search is better suited for solving increasingly complex tasks.

4.3 Typical Behaviour of Resultant Robot Controller

The MASON simulator provides a GUI which allows the experimenter to observe the behaviour of a robot team as controlled by the different networks. When a completely untrained network was used, the robots would mostly spin around on the spot for the duration of the simulation or they would constantly move to the edge of the environment and try move through the wall, ignoring the resources and each other. When a trained network is used, however, the robots seem to move around the environment with a lot more purpose. They move in straight lines and head straight towards construction blocks in order to collect them. On the odd occasion once a team has connected most of the blocks available in the environment, they adopt somewhat of a patrolling movement pattern in order to search the environment, the majority of the individuals circling the boundary of the environment with the odd individual crossing diagonally through the centre.

4.4 Future Work

For this investigation, the hybrid search was implemented as an equally weighted sum of the objective and novelty scores. For future work, it may be of interest to attempt an investigation in which the hybrid search function is implemented using various gradations of novelty and objective. That is to say, instead of calculating the evenly weighted average between the novelty and objective scores, hybrid search could be calculated using various weights for the objective and novelty values.

Devising the behavioural metrics presented a particularly challenging task. Further research into the effects of various metrics is required in order to be able to create an ideal characterisation for novel behaviour.

5. CONCLUSION

This investigation evaluated the ability of objective, novelty and hybrid fitness approaches to evolve multi-robot team controllers using the HyperNEAT EA that were able to solve increasingly complex collective construction tasks.

Construction schemas were used to specify the underlying connection rules of the environment and increase the complexity of the task and the required level of cooperation amongst the robots in a team.

By looking at the results gathered it can be concluded that the search for behavioural novelty is more effective at evolving controllers to solve complex collective construction tasks than both the objective and hybrid functions.

The fact that hybrid search performed better than the objective approach indicates the positive influence that even just a partial novelty search has. That hybrid had also still performed worse than novelty shows the inability of the objective search to deal with the more complex schemas.

It can therefore be concluded that novelty search is able to produce considerably more successful multi-robot team controllers to solve increasingly complex collective construction tasks than objective and hybrid search when using the HyperNEAT evolutionary algorithm.

6. 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] L. Chaimowicz, T. Sugar, V. Kumar, and M. F. M. Campos. An architecture for tightly coupled multi-robot cooperation. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 3, pages 2992–2997. IEEE, 2001.

[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. A novel generative encoding for exploiting neural network sensor and output geometry. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 974–981. ACM, 2007.

[5] 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.

[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] J. Drchal, J. Koutník, and M. Snorek. Hyperneat controlled robots learn how to drive on roads in simulated environment. In *2009 IEEE Congress on*

Evolutionary Computation, pages 1087–1092. IEEE, 2009.

[8] A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

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

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

[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. Robust non-linear control through neuroevolution. 2003.

[16] 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.

[17] 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.

[18] J. Heaton. Encog: Library of interchangeable machine learning models for java. *Journal of Machine Learning Research*, 16:1243–1247, 2015.

[19] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural Networks, 1989. IJCNN., International Joint Conference on*, pages 593–605. IEEE, 1989.

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

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

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

[23] S. Luke, G. Balan, K. Sullivan, and L. Panait. Mason library homepage. http://cs.gmu.edu/ eclab/projects/mason/.

[24] 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.

[25] R. Miikkulainen. Neuroevoluion. In *Encyclopedia of Machine Learning*. Springer, New York, NY, USA, 2010.

[26] D. Murphy. Jbox2d homepage. http://www.jbox2d.org/.

[27] A. L. Nelson, E. Grant, and T. C. Henderson. Evolution of neural controllers for competitive game playing with teams of mobile robots. *Robotics and Autonomous Systems*, 46(3):135–150, 2004.

[28] 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.

[29] S. Risi, J. Lehman, and K. O. Stanley. Evolving the placement and density of neurons in the hyperneat substrate. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 563–570. ACM, 2010.

[30] 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.

[31] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

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

[33] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[34] T. G. van den Berg and S. Whiteson. Critical factors in the performance of hyperneat. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 759–766, New York, NY, USA, 2013. ACM.

[35] T. G. van den Berg and S. Whiteson. Critical factors in the performance of hyperneat. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 759–766. ACM, 2013.

[36] 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.

[37] 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.

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