UNIVERSITY OF CAPE TOWN

MASTERS THESIS

Evolving Morphological Robustness for Collective Robotics

Author:
A. Ruben PUTTER

Supervisor:
Dr. Geoff NITSCHKE

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Research Group Name
Department of Computer Science

October 10, 2019

Signed:

Date:

Contents

Chapter 1

Introduction

1.1 Introduction

The design and development of automated problem solvers and autonomous robots has long been a central theme of research in the crossover between artificial intelligence and robotics [5]. However, given the levels of complexity and non-linearity of real-world problems, traditional methods of controller design have become impractical since they rely mostly on linear mathematical models [11]. An additional consideration is that at a certain point, designing the problem solver becomes more difficult than solving the problem, especially when there is not a distinct and well-defined set of steps to follow in order to obtain the solution.

One such complex task, and the one being focussed on as the subject of this thesis, is that of achieving automated collective behaviour in which a team of autonomous agents must cooperate amongst themselves in order to accomplish a task. More specifically, this thesis will focus on collective robotics with a case study on the collective construction task

While it may be possible to to solve many complex tasks using a single sophisticated robot, there are several advantages to using collective robotics 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 a single point of failure [3]. The downside to a multi-agent team, however, is the infeasibility of designing the intricate dynamics inherent in producing self-organised behaviour between individuals by hand [10].

The collective construction task requires that a team of agents cooperate by coordinating their behaviours in order to assemble various structures within their environment [14]. A group of autonomous agents (robots) are deployed into an unknown environment which they must explore in order to find resources and figure out how to assemble these resources in such a way as to create some structure, which can be completely predefined, or just according to some predefined structural criteria [14].

The ability to automate construction tasks through creating controllers for collective robotics presents several considerable benefits. Automation like this could be used to assist in projects such as producing low-cost housing in locations where resources are scarce, reducing high accident rates due to human error which is unfortunately unavoidable in traditional construction methods [15]. An additional possible application in the near future is to send multi-robot construction teams to extraterrestrial planets in order to build habitable structures in anticipation of humans arriving. These robot teams would also be useful in underwater construction which is extremely difficult and is one of the most dangerous jobs [19].

In an attempt to try and find a solution around the complexity of automating the design of these controllers, researchers turned to biological and natural processes for inspiration.

The problem solving power of evolution is abundantly clear in nature with the diverse range of organisms that exist (or ever have existed) on Earth, each specially designed for optimal survival in its own niche environment [5], making it unsurprising that computer scientists have resorted to researching natural processes for inspiration.

Neuroevolution (NE) is a method of artificially evolving Artificial Neural Networks (ANN's) by implementing Evolutionary Algorithms to perform tasks such as evolving the network connection weights as well as the network topology [16, 20].

One such approach that draws inspiration from Darwinian evolution is Evolutionary Algorithms which relates the fundamental metaphor of the evolutionary process to a simple style of problem solving, which is that of trial-and-error (also referred to as generate-and-test) [5].

It makes use of Evolutionary Algorithms in order to abstract the key aspects of the evolutionary process into a generational loop that will search for an optimal solution out of a population of candidates by evaluating each one in order to determine which is the fittest and thus the characteristics that persist to the next generation. Resulting in a stochastic search for the most optimal solution of the problem space.

==

EDIT 1: ——————— ADD SOME REAL WORLD EXAMPLES AND DESCRIPTIONS OF NE "NE is the main method for controller adaptation in evolutionary robotics (the overall topic that this thesis fits into) - and within this field - NE applied to collective robotics has achieved success (evolved effective group behaviours) to solve various collective behaviour tasks - then give several example tasks that evolutionary collective robotics systems have solved - with references"

==

One of the biggest issues to overcome in order to make these agents as robust as possible (and more suited to their inherently dangerous environments) is determining some way for the robots to be able to continue performing their required tasks despite any physical damage to their physical morphology that they may sustain. More specifically, with regards to their sensory input devices, which are integral to the functioning of any agent.

In fact, an open problem in *collective* [12] and *swarm robotics* [1] is ascertaining appropriate sensory-motor configurations (morphologies) for robots comprising teams that must work collectively given automatically generated controllers [7].

==

EDIT 2: ————————- 'There should also be some mention of indirect vs direct encodings in NE - and mention examples of each (with references) - e.g. NEAT and HyperNEAT - pointing out the benefits of the latter (this is especially important since HyperNEAT is mentioned in your objectives and there needs to be some justification for using it)'

All of the information needed for the above edits (EDIT 1 and 2) is in the previous papers, especially the Honours literature reviews

==

Current robotic systems make use self-diagnosis and selection from pre-designed contingency plans in order to be able to recover from damage and continue its normal functioning [6, 18, 2]. However, robots that implemented this approach of self-diagnosis and recovery proved to be somewhat problematic in that these systems are relatively expensive and are difficult to design since it requires full *a priori* knowledge of the system in order to design the contingency plans [4].

Addressing this, recent work in *Evolutionary Robotics* elucidated the efficacy of population based stochastic trial-and-error methods for online damage recovery in autonomous robots operating in physical environments [4]. This was demonstrated as being akin to self-adaptation and injury recovery of animals observed in nature.

This study further contributes to this research area, focussing on *evolutionary controller design* [7] within the broader context of *collective* [12] and *swarm* [1] robotics. That is, evolutionary controller design for robot groups that must continue to accomplish tasks given that damage is sustained to the morphologies of some or all of the robots in the group.

Given previous non-objective controller evolution work in collective robotics [8] [9] [10], and previous research demonstrating the efficacy of *novelty search* [13] for evolving behaviours that operate across various robotic morphologies, the following research objectives formed the focus of this study;

1. Novelty search behaviours out-perform those of objective serach in terms of average team task performance in increasingly complexy collective construction tasks.

2. Novelty search is suitable for evolving morphologically robust controllers in collective construction tasks.

Another approach to this challenge draws inspiration from self-adaptation and injury recovery that has been observed in animals in nature.

1.1.1 Research Goals

'The aims of this research project can be summarised as follows:'

- 'To investigate the morphological robustness of multi-robot team controllers evolved using the HyperNEAT algorithm with respect sensory configurations when implemented in solving increasingly complex tasks requiring collective behaviour'

- 'building on the first, the second aim is to investigate the three different variations of the HyperNEAT algorithm by implementing various fitness functions'

"The complexity of the collective construction task is controlled by using a construction schema, which is implemented as a set of underlying connection rules. The level of cooperation is controlled by using the different types of resource blocks in the simulation. The various levels are outlined as follows:"

1. Level 1: no complexity or cooperation.

2. Level 2: no complexity and some cooperation.

3. Level 3: some complexity and some cooperation.

We apply the HyperNEAT [17] neuro-evolution method to evolve behavior for a range of morphologically homogenous teams that must solve various collective construction tasks.

This study's research objectives were formulated given results of the following previous related work. First, that non-objective based controller evolution has been effectively demonstrated in collective robotics [10, 8, 9]. Second, that Novelty Search has been demonstrated as suitable for evolving controllers (behaviours) that effectively operate across a range of robot morphologies

1. To demonstrate the efficacy of Novelty Search (as compared to objective-based search) for evolving morphologically robust conrtollers over increasingly complex tasks.

2. To demonstrate the efficacy of Novelty Search evolved behaviours versus those evolved by objective-based search in terms of average task performance (behaviour quality) over increasing task complexity

To test these objectives, experiments evaluated various robot morphologies in an increasing complex collective construction task. That is, morphological robustness of evolved controllers was evaluated in terms of a controller's task performance coupled with alternate robot morphologies.

This study's contribution was thus to elucidate the impact of specific objective versus non-objective search methods on the evolution of *morphologically robust* controllers in collective robotic systems. To date, the morphological robustness of such controller evolution approaches has not been comparatively evaluated, especially in the context of collective robotics. Specifically, collective robotic systems that must adapt to unforeseen morphological change, such as the loss or damage of sensors on one of more robots without significant task performance degradation [2] [4]

1.1.2 Contributions

1.1.3 Scope

This dissertation is mainly focussed on exploring the ability of the HyperNEAT algorithm to produce homogenous multi-agent ANN controllers which are robust to changes in their sensory input morphology. That is to say, whether or not a team of evolved controllers/agents are able to still accomplish the task for which they were trained despite having sustained damage to their input sensors in some way.

Due to the large variety of approaches and parameter tuning abilities when it comes to Neuroevolution methods, we have chosen to focus specifically on the following search functions combined with indirect encoding method of NEAT, called HyperNEAT:

- Objective Search
- Novelty Search
- Hybrid Search

Additionally, these experiments will be implemented using a predetermined collection of sensors that are either arranged in different physical locations on the robot or disabled to simulate damage. The two distinctive facets of the evaluations is that the controllers are re-implemented using different morphologies as well as an additional evaluation where the sensors are randomly activated/deactivated.

1.1.4 Structure

This dissertation has been structured in the following sections

- Introduction: A brief outline of the research and related work as well as some of the applications and benefits of this research.

- Related Research: An in-depth investigation into the various main topics incorporated in this investigation (Machine Learning, Evolutionary Algorithms/Computing, Multi-Agent tasks).

- Methods: An outline of the various tools and approaches that were used in order to be able to conduct this research. This includes the simulator that was to simulate the environment, the machine learning library used to implement the HyperNEAT algorithm, the sensors and robot types that were simulated, as well as different morphologies and calculations for the fitness functions.
- Results: The results of the different experiments that were conducted. This includes the graphs and descriptions of what the different metrics mean and how they are interpreted.
- Conclusion: A section in which we discuss the results shown in the previous section and how they relate to the research goals and objectives.

Chapter 2

Background Research

2.1 Robustness

Currently, robotic systems recover from damage via self diagnosis and selection from pre-designed contingency plans in order to be able to continue functioning (check the references for this from the GECCO Penultimate paper). However, robots using such self-diagnosis and recovery systems are problematic as such systems are expensive, require sophisticated monitoring sensors and are difficult to design since system designers must have a priori konwledge of all necessary contingency plans (also get a reference).

2.2 Artifical Neural Networks

'Artificial neural networks are computational methodologies that perform multifactorial analyses. Inspired by networks of biological neurons, artificial neural network models contain layers of simple computing nodes that operate as nonlinear summing devices. These nodes are richly interconnected by weighted connection lines, and the weights are adjusted when data are presented to the network during a "training" process. Successful training can result in artificial neural networks that perform tasks such as predicting an output value, classifying an object, approximating a function, recognizing a pattern in multifactorial data, and completing a known pattern ' [**dayhoff2001artificial**].

'Artificial neural netowrks have been the subject of an active field of research that has matured greatly over the past 40 years. The first computational trainable neural networks were developed in 1959 by Rosenblatt as well as by Widrow and Hoff and Widrow and Stearns [refs 1-3]. Rosenblatt perceptron was a neural network with 2 layers of computational nodes and a single layer of interconnections. It was limited to the solution of linear problems. For example, in a two-dimensional grid on which two different types of points are plotted, a perceptron could divide those points only with a straight line; a curve was not possible. Whereas using a line is a linear discrimination, using a curve is a non-linear task. Many problems in discrimination and analysis cannot be solved by a linear capability alone.' [**dayhoff2001artificial**].

2.2.1 Architectures

An Aritificial Neural Network (ANN) is an abstracted and simplified model that represents the functioning of a biological brain [**mcculloch1943logical**]. A single ANN consists of numerous interconnected simple computational units, called neurons, that are ordered into layers so as to create a neural net [**RefWorks:31**].

An ANN consists of a set of processing elements, also known as neurons or nodes, which are interconnected [20].

An ANN can be described as a directed graph in which each node, i, performs a transfer function f_i of the form [20] (this is the sentence that originally introduced the function outlined here **??**)

Each neuron consists of several inputs, some specific activation function and some output. An ANN receives input from the environment at its input layer [11].

Each neuron in a non-input layer then calculates the weight-ed sum of its inputs, referred to as the activation value [**yegnanarayana2009artificial**, **RefWorks:31**], 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 [**yegnanarayana2009artificial**].

The operational/functioning of a single neuron can be explained using the following equation:

$$y_i = f_i(\sum_{j=1}^{n} w_{ij}x_j - \theta_i) \tag{2.1}$$

can maybe combine the above equation with a simple diagram of an ANN so that it would be easier to understand what is going on in the equation, can provide a better explanation of the different components in the equation and the relationships between them

consisting of the following components:

- y_i is the output that is produced by the i^{th} neuron in the network.
- w_{ij} is the weighted value of the connection between the i^{th} and j^{th} nodes in the network.
- x_j is the j^{th} input received by the neuron.
- θ_i is the threshold value (or bias) of the node. can maybe find a more detailed description of how this bias works or how it is used and in what use case examples
- f_i is the transfer function/activation function performed by the node when it receives the input from the input-noes. This function is usually nonlinear, such as one of the following: remember to try and get some examples of the following function types
  - Heavyside
  - Sigmoid
  - Gaussian

In **??**, each term in the summation only involves one input x_j. Higher-order ANN's are those that contain high-order nodes, i.e., nodes in which more than one input are involved in some of the terms of the summation. For example, a second-order node can be described as:

$$y_i = f_i(\sum_{j,k=1} w_{ijk}x_jx_k - \theta_i) \tag{2.2}$$

where all the components have similar definitions to those outlined in **??**.

above is the equation for a function that receives input from more than one preceding nodes

ANNs are universal function approximators [find a reference for this statement]. It has been shown that a network can approximate any continuous function to any desired accuracy [**zhang1998forecasting**]. This makes ANNs a favourable choice for agent controllers in accomplishing various tasks [**yegnanarayana2009artificial**].

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 [**dayhoff2001artificial**, **RefWorks:1**].

"The architecture of an ANN is determined by its topological structure, i.e., the overall connectivity and transfer function of each node in the network" [20]

there should be another subsection on the structure of ANN's?

2.2.2 Transfer Functions

Can maybe have a section dedicated to the different types of functions that are used in the Neurons. Sigmoid, Gaussian

2.3 Learning in ANN's

"The variety and complexity of learning systems makes it difficult to formulate a universally accepted definition of learning. However, a common denominator of most learning systems is their capability for making structural changes to themselves over time with the intent of improving performance on tasks defined by their environment, discovering and subsequently exploiting interesting concepts, or improving the consistency and generality of internal knowledge structures " [**de1988learning**].

"Given this perspective, one of the most important means for understanding the strengths and limitations of a particular learning system is a precise characterization of the structural changes that are permitted and how such changes are made " [**de1988learning**] THIS IS A GOOD EXPLANATION OF HOW THE DIFFERENT LEARNING APPROACHES ARE CHARACTERIZED. THIS WILL BE A GOOD INTRODUCTION IN TO THE LATER SECTIONS ON THE ALGORITHMS AND WHAT MAKES THEM DIFFERENT TO EACH OTHER "In classical terms, this corresponds to a clear understanding of the space of possible structural changes and the legal operators for selecting and making changes. This perspective also lets one more precisely state the goal of the research in applying genetic algorithms to machine learning, namely, to understand when and how genetic algorithms can be used to explore spaces of legal structural changes in a goal-oriented manner. " [**de1988learning**]. BE SURE TO STRUCTURE THE CONTENT SUCH THAT IT ALIGNS WITH THE WAY IN WHICH RESEARCH DEVELOPED. FIRST ANNS THEN LEARNING THEN GA THEN EC etc.

what is learning "An agent is learning if it improves its performance on future tasks after making observations about the world." [**russell2016artificial**]

"Why would we want an agent to learn? If the design of the agent can be improved, why wouldn't the designers just program in that improvement to begin with? There are three main reasons. First, the designers cannot anticipate all possible situations that the agent might find itself in. For example, a robot designed to navigate mazes must learn the layout of ecah new maze it encounters." [**russell2016artificial**] "Second, the designers cannot anticipate all changes over time; a program designed to predict tomorrow's stock market prices must learn to adapt when conditions change from boom to bust" [**russell2016artificial**]. "Third, somethimes human programmers have no idea how to program the solution themselves. For example, most people are good ate recognizing the faces of family members, but even the best programmers are unable to program a computer to accomplish that task, except by using learning algorithms" [**russell2016artificial**].

'Any component of an agent can be improved by learning from data. The improvements, and the techniques used to make them, depend on four major factors:' -¿ 'Which *component* is to be improved.' -¿ 'What *prior knowledge* the agent already has' -¿ 'What *representation* is used for the data and the component.' -¿ 'What *feedback* is available to learn from.'

'Components to be learned: ' [**russell2016artificial**] remember to rephrase everything listed below The components of these agents include: 1. A direct mapping from conditions on the current state to actions. 2. A means to infer relevant properties of the world from the percept sequence (can maybe check the reference to see how to describe this) 3. Information about the way the world evolves and about the results of possible actions the agent can take. 4. Utility information indicating the desirability of world states. 5. Action-value information indicating the desirability of actions. 6. Goals that describe classes of states whose achievement maximizes the agent's utility.

'Each of these components can be learned. Consider, for example, and agent training to become a taxi driver. Every time the instructor shouts "BRAKE!" the agent might learn a condition-action rule for when to brake (component 1); the agent also learns every time the instructor does not shout.' [**russell2016artificial**].

'By seeing many camera images that it is told contain buses, it cn learn to recognize them (2)' [**russell2016artificial**].

'By trying actions and observing the results - for example, braking hard on a wet road - it can learn the effects of its actions (3)' [**russell2016artificial**].

'Then, when it receives no tip from passengers who have been thoroughly shaken up during the trip, it can learn a useful component of its overall utility function (5)' [**russell2016artificial**]

there are no examples for the last 2 components...? check similar resources to find this

'There are 3 types of feedback that determine the three main types of learning: ' [**russell2016artificial**] -¿ 'Unsupervised Learning: the agent learns patterns in the input even though no explicit feedback is supplied. The most common unsupervised learning task is clustering, detecting potentiall useful clusters of input examples. For example, a taxi agent gradually develop a concept of "good traffic days" and "bad traffic days" without ever being given labelled examples of each by a teacher' -¿ 'In Reinforcement Learning the agent learns from a series of reinforcements - rewards or punishments, For example, the lack of a tip at the end of a journey gives the taxi agent an indication that it did something wrong. The to points for a win at the end of a chess game tells the agent it did something right. It is up to the agent to decide which of the actions prior to the reinforcement were most responsible for it' -¿ 'In supervised learning the agent observes some example input-output pairs and learns a function that maps from input to output. In component 1 above, the inputs are percepts and the output are provided by a teacher who says "Brake" or "Turn Left". In component 2, the inputs are camera images and the outputs again come from a teacher who says "that is a bus". In component 3, the theory of braking is a function from states and braking actions to stopping distance in feet. In this case, the output value is available directly from the agents percepts (after the fact); the environment is the teacher'

"Never tell people how to do things. Tell them what to do and they will surprise you with their ingenuity" - George S. Patton

Learning in ANN's, also referred to as 'training', is typically achieved using examples. During this process, the connection weights between neurons in the network are iteratively adjusted until such a point that the network can perform the desired task [20].

Learning in ANN's can roughly be divided into supervised, unsupervised, and Reinforcement Learning.

- 'Supervised Learning is based on direct comparison between the actual output of an ANN and the desired correct output, also known as the target output. It is often formulated as the minimization of an error function, such as the total mean square error between the actual output and the desired output, summed over all available data. A gradient descent-based optimization algorithm can then be used to adjust connection weights in the ANN iteratively in order to minimize the error'

- Reinforcement Learning is a special case of Supervised Learning where the exact desired output is unkinown. It is based only on the information of whether or not the actual output is correct.

- Unsupervised Learning is solely based on the correlations among input data. No information on "correct output" is available for learning.

At the core of the learning algorithm is the 'learning rule', which is what determines the manner in which the connection weights are adapted during the iterative learning process [20].

Some examples of popular learning rules include:

- Delta Rule

- Hebbian Rule

- Anti-Hebbian Rule

- Competitive Learning Rule

A typical cycle of the evolution of connection weights [20]:

1. Decode each individual (genotype) in the current generation into a set of connection weights and construct a corresponding ANN with the weights.

2. Evaluate each ANN by computing its total mean square error between actual and target outputs (Other error functions ma yalso be used). The fitness of an individual is determined by the error. The higher the error, the lower the fitness (because you are essentially measuring how close the actual behaviour is to the desired behaviour.). The optimal mapping from the error to the fitness is problem dependent. A regularization term may be included in the fitness function to penalize large weights

3. Select parents for reproduction based on their fitness.

4. Apply search operators, such as crossover and/or mutation, to parents in order to generate offspring, which will form the next generation.

2.3.1 Learning Rules

In this section, we will investigate and elaborate on the details and various implementations of the learning rules that were outlined above.

'An artificial neural networks learning rule or learning process is a method, mathematical logic or algorithm which improves the networks performance and/or training time. Usually, this rule is applied repeatedly over the network. It is done by updating the weights and bias levels of a network when a network is simulated in a specific data environment. A learning rule may accept existing conditions (weights and biases) of the network and will compare the expected result and actualresult of the network to give new and improved values for weights and bias. Depending on the complexity of actual model being simulated, the learning rule of the network can be as simple as an XOR logic gate or mean squared error, or as complex as the result of a system of differential equations.' 'The learning rule is one of the factors which decides how fast or how accurately the artificial network can be developed. Depending on the process to develop the network there are three main models of machine learning: (1) Unsupervised, (2) Supervised, (3) Reinforcement'

2.3.2 Supervised Learning

page 694-695 of the cited text

'The task of supervised learning is this: ' [**russell2016artificial**] (everything below here is copied from the cited source, remember to reword) (This is the training set) Given a training set of N example input-output pairs (x_1, y_1), (x_2, y_2), (x_3, y_3),...,(x_N, y_N) where each y_i was generated by an unknown function $y = f(x)$, discover a function h that approximates the true function f. (This is the Hypothesis) Here x and y can be any value; they need not be numbers. The function h is a hypothesis. Learning is a search through the space of possible hypotheses for one that will perform well, even on new examples beyond the training set. To measure the accuracy of a hypothesis we give it a **test set** of examples that are distinct from the training set. We say a hypothesis **generalizes** well if it correctly predicts the value of y for novel examples. Sometimes the function f is stochastic - it is not strictly a function of x, and what we have to learn is a conditional probability distribution, $P(Y|x)$. (Classification) When the output y is one of a finite set of values (such as *sunny*, *cloudy*, or *rainy*), the learning problem is called **classification**, and is called Boolean or binary classification if there are only 2 values. (Regression) Wehn y is a number (such as tomorrow's temperature), the learning problem is called **regression**. (Technically, solving a regression problem is finding a conditional expectation or average value of y, because the probability that we have found *exactly* the right real-valued number for y is 0.)

Backpropagation

"Backpropagation is a method used in ANNs to calculate a gradient that is needed in the calculation of the weights to be used in the network. Backpropagation is shorthand for 'backward propagation of errors', since an error is computed at the output and distributed backwards throughout the network's layers."

"Backpropagation is a generalization of the Delta Rule to multi-layered feedforward networks, made possible by using the chain rule to iteratively compute gradients for each layer."

"Backpropagation is a special case of a more general technique called automatic differentiation. In the context of learning, backpropagation is commonly used by the gradient descent optimization algorithm to adjust the weights of neurons by calculating the gradient of the loss function"

Gradient Descent

"Gradient descent is a first order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point. If, instead, one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is known as gradient ascent"

'Predict a continuous target variable'

2.4 Unsupervised Learning

2.5 Reinforcement Learning

can maybe have a subsection for the different types of RL techniques such as: -¿ Value and Policy Iteration -¿ Q-learning

'Reinforcement Learning dates all the way back to the early days of cybernetics and work in statistics, psychology, neuroscience and computer science.'[**KaelblingLittmanMoore1996**] 'In the last 5 - 10 years it has attracted rapidly increasing interest in the machine learning and atificial intelligence communities. Its promise is beguiling - a way of programming agents by reward and punishment without needing to specify how the task is to be achieved' [**KaelblingLittmanMoore1996**] -¿ This could be a good line to have in the introduction section, how scientists are turning to automated methods of designing controllers. 'Reinforcement Learning is the problem faced by an agent that must learn behaviour through trial-and-error interactions with a dynamic environment' [**KaelblingLittmanMoore1996**]. 'The work described here has a strong family resemblance to eponymous work in psychology, but differs considerably in the details ad in the use of the word "reinforcement". It is appropriately thought of as class of problems, rather than as a set of techniques' [**KaelblingLittmanMoore1996**]. While the approach take in Reinforcement Learning bears a strong resemblance to eponymous work in psychology, it is different in its use of the word 'reinforcement' and can be thought of as a class of problems instead of a set of techniques [**KaelblingLittmanMoore1996**].

'There are two main strategies for solving reinforcement learning problems:" -¿ 'The first is to search in the space of behaviours in order to find one that performs well in the environment. This approach has been taken by work in genetic algorithms and genetic programming' what is the difference between genetic algorithms and genetic programming? -¿ 'The second is to use statistical techniques and dynamic programming methods to estimate the utility of taking actions in states of the world. This paper is devoted almost entirely to the second set of techniques because they take advantage of the special structure of reinforcement learning problems that is not available in optimization problems in general' [**KaelblingLittmanMoore1996**]

'In the standard Reinforcement Learning model, an agent is connected to its environment via perception and action (there is a figure in the source paper). On each step of interaction the agent receives as input, i, some indication of the current state, s, of the environment; the agent then chooses an action, a, to generate as output. The action changes the state ofthe environment, and the value of this state transition is communicated to the agent through a scalar reinforcment signal, r. The agents behaviour, B, should choose actions that tend to increase the long-run sum of values of the reinforcement signal. It can learn to do this over time by a systematic trial-and-error, guided by a wide variety of algorithms that are the subject of later sections in this paper' This variety of algorithms, are they referring to the learning rules?

A Reinforcement Learning model formally consists of [**KaelblingLittmanMoore1996**]:

- A discrete set of environment states, S.
- A discrete set of agent actions, A.
- A set of scalar reinforcement signals; typically 0,1, or the real numbers.

'The figure also includes an input function I, which determines how the agent views the environment state; we will assume that it is the identity function (that is, the agent perceives the exact state of the environment) until we consider partial observability in Section 7' [**KaelblingLittmanMoore1996**]

An intuitive way to understand the relation between the agent and its environment is with the following example dialogue: Environment: "You are in state 65. You have 4 possible actions" Agent: "I'll perform action 2" Environment: "You received a reinforcement of 7 units. You are now in state 15. You have 2 possible actions" Agent: "I'll perform action 1" Environment: "You received a reinforcement of -4 units. You are now in state 65. You have 4 possible actions" Agent: "I'll perform action 2" Environment: "You received a reinforcement of 5 units. You are now in state 44. You have 5 possible actions" etc.

The agent 'remembers' the results of the actions that it performs. It knows which action in which state produced the highest reinforcement value.

'The agents job is to find a policy π, mapping states to actions, that maximizes some long-run measure of reinforcement. We expect, in general, that the environment will be non-deterministic; that is, that taking the same action in the same state on two different occasions may result in different next states and/or different reinforcement values. This happens in the example above: from state 65, applying action 2 produces differing reinforcements and differing states on two occasions. However, we assume the environment is stationary; that is, that the probabilities of making state transitions or receiving specific reinforcement signals do not change over time ' [**KaelblingLittmanMoore1996**]

'Reinforcement Learning differes from the more widely studied problem of supervised learning in several ways. The most important difference is that there is no presentation of input/output pairs. Instead, after choosing an action the agent is told the immediate reward and the subsequent state, but it is NOT told which action would have been in its best long-term interests. It is necessary for the agent to gather useful experience about the possible system states, actions, transitions and rewards actively to act optimally. Another difference from supervised learning is that on-line performance is important: the evaluation of the system is often concurrent with learning' [**KaelblingLittmanMoore1996**]

'Some aspects of reinforcement learning are closely related to search and planning issues in artificial intelligence. AI search algorithms generate a satisfactory trajectory through a graph of states. Planning operates in a similar manner, but typicall within a construct with more complexity than a graph, in which states are represented by compositions of logical expressions instead of atomic symbols. These AI algorithms are less general than the reinforcement learning methods, in that they require a predefined model of state transitions, and with a few exceptions assume determinism.

On the other hand, RL, at least in the kind of discrete cases for which theory has been developed, assumes that the entire state space can be enumerated and stored in memory - an assumption to which conventional search algorithms are not tied.' [**KaelblingLittmanMoore1996**]

Models of Optimal Behaviour 'Before we can start thinking about algorithms for learning to behave optimally, we have to decide what our model of optimality will be. In particular, we have to specify how the agent should take the future into account in the decisions it makes about how to behave now. There are three models that have been the subject of the majority of work in this area' [**KaelblingLittmanMoore1996**].

The Finite Horizon Model

[**KaelblingLittmanMoore1996**] 'this is the easiest one to think about' (conceptualize?) 'at a given moment in time, the agent should optimize its expected reward for the next h steps:'

$$E(\sum_{t=0}^{h} r_t) \tag{2.3}$$

2.6 Genetic Algorithms

need to find an appropriate place for this section w.r.t the rest of the research. GA are the basis of NE algorithms. Since learning methods are classified based on their representations, NE makes use of genetic encodings right?

WHAT IS THE DIFFERENCE BETWEEN A GENETIC ALGORITHM AND EVOLUTIONARY ALGORITHM? A genetic algorithm is a class evolutionary algorithm. Although genetic algorithms are the most frequently encountered type of EA, there are other types such as Evolution Strategy https://stackoverflow.com/questions/2890061/what-is-the-difference-between-genetic-and-evolutionary-algorithms These algorithms are defined by the way in which the agents are represented. Genetic Algorithms make use of binary strings (I think???)

'Simply stated, genetic algorithms are probabilistic search procedures designed to work on large spaces involving states that can be represented by strings. These methods are inherently parallel, using a distributed set of examples from the space (population of strings) to generate a new set of samples. They also exhibit a more subtle implicit parallelism. Roughly, in processing a population of m strings, a genetic algorithm implicitly evaluates substantially more than m^3 component substrings. It the automatically biases future populations to exploit the above average components as building blocks from which to construct structures that will exploit regularities in the environment (problem space)" [**goldberg1988genetic**]

'The theorem that establishes this speedup and its precursors - the schema theorems - illustrate the central role of theory in the development of Genetic Algorithms. Learning programs designed to exploit this building block property gain a substantial advantage in complex spaces where they must discover both the "rules of the game" and the strategies for playing the "game"' [**goldberg1988genetic**].

should probably just make sure of the content that I am using from [**goldberg1988genetic**] since the paper was published 30 years ago

"Although there are a number of different types of genetics-based machine learning systems, in this issue we concentrate on classifier systems and their derivatives. Classifier systems are parallel production systems that have been designed to exploit the implicit parallelism of genetic algorithms. All interactions are via standardized messages, so that conditions are simply defined in terms of the messages they send. The resulting systems are computationally complete, and the simple syntax makes it easy for a GA to discover building blocks appropriate for the construction of new candidate rules. Because classifier systems rely on competition to resolve conflicts, the need no algorithms for determining the global consistency of a set of rules. As a consequence, new rules can be inserted into an existing system, as trials or hypotheses, without disturbing established capacities. This gracefulness makes it possible for the system to operate incrementally, testing new structures and hypotheses while steadily improving its performance"

"Genetic Algorithms are a family of adaptive search procedures that have been described and extensively analyzed in the literature" [**de1988learning**] (just use the family of procedures part)

"GA's derive their nam from the fact that they are loosely based on models of genetic change in a population of individuals" [**de1988learning**].

"These models consist of three basic elements: " [**de1988learning**] (1) 'a Darwinian notion of "fitness", which governs the extent to which an individual can influence future generations' (2) 'a "mating operator", which produces offspring for the next generation' (3) '"genetic operators," which determine the genetic makeup of offspring from the genetic makeup of the parents'

what is the difference between GA and EA??

"A key point of these models is that adaptation proceeds, not by making incremental changes to a single structure, but by maintaining a population (or database) of structures from which new structures are created using genetic operators such as crossover and mutation. Each structure in the population has an associated fitness (goal-oriented evaluation), and these scores are used in a competition to determine which structures are used to form new ones" [**de1988learning**].

"The key feature of a GA's is their ability to exploit accumulating information about an initially unknown search space in order to bias subsequent search into useful subspaces. Clearly, if one has a strong domain theory to guide the process of structural change, one would be foolish not to use it. However, for many practical domains of application, it is very difficult to construct such theories. If the space of legal structual changes is not too large, one can usuall develop an enumerative search strategy with appropriate heuristic cutoffs to keep the computation time under control. If the search space is large, however, a good deal of time and effort can be spent in developing domain-specific heuristics with sufficient cutoff power. It is precisely in these circumstances (large, complex, poorly understood search spaces) that one should consider exploiting the power of genetic algorithms" [**de1988learning**].

"At the same time, one must understand the price to be paid for searching poorly understood spaces. It typically requires 500-1000 samples before genetic algorithms have sufficient information to strongly bias subsequent samples into useful subspaces. This means that GA's will not be appropriate searcch procedures for learning domains in which the evaluation of 500-1000 alternative structural changes is infeasible. The variety of current activity in using GAs for machine learning suggests that many interesting learning problems fall into this category" [**de1988learning**].

"A simple and intuitive approach for effecting behavioural changes in a performance system is to identify a key set of parameters that control the system's behaviour, and to develop a strategy for changing those parameters values to improve performance. The primary advantage of this approach is that it immediately places us on the familiar terrain of parameter optimization problems, for which there is considerable understanding and guidance, and for which the simplest forms of GAs can be used. It is easy at first glance to discard this approach as trivial and not at all representative of what is meant by 'learning'. But note that significant behavioural changes can be achieved within this simple framework."

2.7 Evolutionary Computing / Evolutionary Algorithms

for the basic outline below, this was taken directly from the outline in [20], but I have added an additional first step that mentnions letting the individuals in the population first attempt the task in the simulated environment. This is so that there are actually some results to analyse as part of the first step. The original one does not mention what the individuals do to 'prove' their fitness or how the fitness would be evaluated.

try to find another reference where they have a general outline of an Evolutionary Algorithm. You can then reference both of these for the outline that you have created below.

Basic outline of evolutionary algorithm

1. Generate the initial population $G(0)$ at random, and set $i = 0$, where i refers to the initial timestamp/first generation check this understanding is correct
2. REPEAT:
   (a) Let each individual run in the simulation so as to be able to view their performance.
   (b) Evaluate the performance of each of the individuals using the predetermined fitness function.
   (c) Select the parents from G_i based on their fitness value (calculated using the fitness function).
   (d) Apply search operators be sure to include some more information on these search operators and the different ones that exist to the parent generation in order to produce the offspring which form $G(i + 1)$
   (e) Advance the timestep maybe rephrase this $i = i + 1$
3. UNTIL *terminationcriterion* is satisfied.

in our case, the termination criterion is a set number of generations for the Evolutionary Algorithm to be performed

2.8 Difficulties to overcome

Not sure about the label of this section Basically have an entire research section elaborating on the trade-offs that need to be made in order to implement the various algorithms

2.8.1 Exploitation vs Exploration

REINFORCEMENT LEARNING PAPER [**KaelblingLittmanMoore1996**] 'One major difference between reinforcement learning and supervised learning is that a reinforcement learner must explicitly explore its environment.' [**KaelblingLittmanMoore1996**] This more refers to the fact that the agent learns through trial-and-error and that they will need to explore the possible steps/actions/solutions and 'remember' the result of that action. (Does it also somehow refer to the mappings between actions and consequences?)

'In order to highlight the problems of exploration, we treat a very simple case in this section. The fundamental issues and approaches described here will, in many cases, transfer to the more complex instances of reinforcement learning discussed later in the paper' [**KaelblingLittmanMoore1996**]

'The simplest possible Reinforcement Learning problem is known as the $k-armed$ bandit problem' [**KaelblingLittmanMoore1996**] 'The agent is in a room with a collection of k gambling machines. The agent is permitted a fixed number of pulls, h. Any arm may be pulled on each turn. The machines do not require a deposit to play; the only cost is wasting a pull playing a suboptimal machine. When arm i is pulled, machine i pays off 1 or 0, according to some underlying probability parameter p_i, where payoffs are independent events and the p_is are unknown. What should the agents strategy be?'[**KaelblingLittmanMoore1996**]. 'This problem illustrates the fundamental tradeoff between exploitation and exploration. The agent might believe that a particular arm has a fairly high payoff probability; should it choose that arm all the time, or should it choose another one that it has less information about, but seems to be worse? Answeres to these questions depend on how long the agent is expected to play the game; the longer the game lasts, the worse the consequences or prematurely converging on a sub-optimal arm, and the more the agent should explore'

2.9 Neuroevolution

find some more information regarding the natural processes: like the phenotype and genotype and why the encoding is so useful for representing regularities

'The primary motivation for neuroevolution is to be able to train neural networks in sequentual decision tasks with sparse reinforcement information. Most neural network learning is concerned with supervised tasks, where the desired behaviour is described in a corpus of input-output examples.' [**Miikkulainen2010**]

The main benefit of neuroevolution compared to other reinforcement learning methods in such tasks is that it allows representing continuous state and action spaces and disambiguating hidden states naturally. Network activations are continuous, and the network generalizes well between continuous values, largely avoiding the state explosion problem that plagues many reinforcement-learning approaches. Recurrent networks can encode memories of past states and actions, making it possible to learn in partially observable Markov Decision Process (POMDP) environments that are difficult for many RL approaches [**Miikkulainen2010**].

Compared to other neural network learning methods, neuroevolution is highly general. As long as the performance of the networks can be evaluated over time, and the behaviour of the network can be modified through evolution, it can be applied to a wide range of network architectures, including those with non-differentiable activation functions and recurrent higher-order connections. While most neural learning algorithms focus on modifying the weights only, neuroevolution can be used to optimize other aspects of the networks as well, including activation functions and network topologies.

Third, neuroevolution allows combining evolution over a population of solutions with lifetime learning in individual solutions: the evolved networks can each learn further through eg. backpropagation or Hebbian learning. The approach is therefore well suited for understanding biological adaptation, and for building artificial life systems.

2.9.1 Basic Methods

(Remember to reword this section below) 'In neuroevolution, a population of genetic encodings of neural networks is evolved in order to find a network that solves the given task. Most neuroevolution methods follow the usual generate-and-test loop of evolutionary algorithms' [**Miikkulainen2010**]

'Each encoding in the population (a genotype) is chosen in turn and decoded into the corresponding neural network (a phenotype). This network is then employed in the task, and its performance over time measured, obtaining a fitness value for the corresponding genotype. After all members of the population have been evaluated in this manner, genetic operators are used to create the next generation of the population. Those encodings with the highest fitness are mutated and crossed over with each other, and the resulting offspring replaces the genotypes with the lowest fitness in the population. The process therefore constitutes an intelligent parallel search towards better genotypes, and continues until a network with a sufficiently high fitness is found' [**Miikkulainen2010**].

this information leads on nicely from how learning is described as being a search through a space of possible solutions. In this case, each ANN is a possible solution function in the defined problem space. There is some way of explaining this in your own words

'Several methods exist for evolving neural networks depending on how the networks are encoded. The most straightforward encoding, sometimes called Conventional Neuroevolution (CNE), is formed by concatenating the numerical values for the network weights (either binary or floating point) [refs 5,16,21 from the original paper introduction]. This encoding allows evolution to optimize the weights of a fixed neural network architecture, an approach that is easy to implement and practical in many domains.' [**Miikkulainen2010**].

'In more challenging domains, the CNE approach suffers from three problems: (1) The method may cause the population to converge before a solution is found, making further progress difficult (i.e premature convergence); (2) similar networks, such as those where the order of nodes is different, may have different encodings, and much effort is wasted in trying to optimize them in parallel (i.e. competing conventions); (3) a large number of parameters need to be optimized at once, which is difficult through evolution' [**Miikkulainen2010**].

check if any of the below texts are better suited to be placed in the section for Learning in ANNs (this is based on the section heading from the original text)

'More sophisticated encodings have been devised to alleviate these problems. One approach is to run the evolution at the level off solution components instead of full solutions. That is, instead of a population of complete neural networks, a population of network fragments, neurons, or connection weights is evolved [refs 7, 13, 15 of original paper]. Each individual is evaluated as part of a full network, and its fitness reflects how well it cooperates with other individuals in forming a ful network. Specifications for how to combine the components into a full network can be evolved separately, or the combinarion can be based on designated roles for subpopulations. In this manner, the complex problem of finding a solution network is broken into several smaller subproblems; evolution is forced to maintain diverse solutions, and competing conventions and the number of parameters is drastically reduced' [**Miikkulainen2010**].

'Another approach is to evolve the network topology, in addition to the weights. The idea is that topology can have a large effect on function, and evolving appropriate topologies can achieve good performance faster than evolving weights only [refs 2, 5, 18, 21 from original paper]. Since topologies are explicitly specified, competing conventions are largely avoided. It is also possible to start evolution with simple solutions and gradually make them more complex, a process that takes place in biology and is a powerful approach in machine learning in general. Speciation according to the topology can be used to avoid premature convergence, and to protect novel topological solutions until their weights have been sufficiently optimized.' [**Miikkulainen2010**].

'All of the above methods map the genetic encoding directly to the corresponding neural network, i.e. each part of the encoding corresponds to a part of the network, and vice versa. Indirect encoding, in contrast, specifies a process through which the network is constructed, such as cell division or generation through a grammar [refs 5, 8, 17, 21 of original paper]. Such an encoding can be highly compact, and also take advantage of modular solutions. The same structures can be repeated with minor modifications, as they often are in biology. It is, however, difficult to optimize solutions produced by indirect encoding, and realizing its full potential is still future work.' [**Miikkulainen2010**].

'The fifth approach is to evolve an ensemble of neural networks to solve the task together, instead of a single network [ref 11]. This approach takes advantage of the diversity in the population: different networks learn different parts or aspects of the training data, and together the whole ensemble can perform better than a single network. Diversity can be created through speciation and negative correlation, encouraging useful specializations to emerge. The approach can be used to design ensembles for classification problems, but it can also be extended to control tasks ' [**Miikkulainen2010**].

what is the difference between classification, regression, and control tasks? are these the only different options? Check if there are any different approaches that you may have missed out on

2.9.2 Applications

this is super useful information because it says that if the network can solve a problem, that it can then solve the more complex version of the same problem, which directly relates to the investigation being conducted in this paper

'Neuroevolution methods are powerful especially in continuous domains of reinforcement learning, and those that have partially observable states. For instance in the benchmark task of balancing the inverted pendulum without velocity information (making the problem partially observable), the advanced methods have been shown to find solutions two orders of magnitude faster than value-function based reinforcement learning methods (measured by number of evaluations [ref 7]. They can also solve harder versions of the problem, such as balancing two poles simultaneously'

'The method is powerful enough to make any real-world applications of reinforcement learning possible. The most obvious area is adaptive, nonlinear control of physical devices. For instance, neural network controllers have been evolved to drive mobile robots, automobiles, and even rockets [refs 6, 14, 19]. The control approach have been extended to optimize systems such as chemical processes, manufacturing systems, and computer systems. A crucial limitation with current approaches is that the controllers usually need to be developed in simulation and transferred to the real system. Evolution is strongest as an off-line learning method where it is free to explore potential solutions in parallel ' [**Miikkulainen2010**].

'Evolution of neural networks is a natural tool for problems in artificial life. Because networks implement behaviours, it is possible to design neuroevolution experiments on how behaviours such as foraging, pursuit and evasion, hunting and herding, and even communication may emerge in response to environmental pressure [ref 20] ' [**Miikkulainen2010**].

'It is possible to analyze the evolved circuits and understand how they map to function, leading to insights into biological networks [ref 10]. The evolutionary behaviour approach is also useful for constructing characters in artificial environments, such as games and simulators. Non-player characters in current video games are usuall scripted and limited; neuroevolution can be used to evolve complex behaviours for them, and even adapt them in real time [ref 12] ' [**Miikkulainen2010**].

Definition

'Neuroevolution is a method for modifying neural network weights, topologies, or ensembles in order to learn a specific task. Evolutionary computation is used to search for network parameters that maximize a fitness function that measures performance in the task. Compared to other neural network learning methods, neuroevolution is highly general, allowing learning withou explicit targets, with nondifferentiable activation functions, and with recurrent networks. It can also be combined with a standard neural network learning (eg. model biological adaptation.). Neuroevolution can also be seen as a policy search method for reinforcement learning problems, where it is well suited to continuous domains and to domains where the state is only partially observable' [**Miikkulainen2010**].

FROM THE LITERATURE REVIEW VERBATIM

Neuroevolution (NE) provides a way of combining EAs and ANNs [**RefWorks:31**]. It uses EAs to evolve the connection weights, topologies and activation functions of ANNs in order to learn a specific task or behaviour [**gomez1999solving**]. NE searches for an ANN with optimal performance based on a fitness function that determines an ANN's suitability to performing a task [**RefWorks:31**].

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 [11].

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 [**gomez2001neuro**, **RefWorks:1**].

In a standard RL scenario an ANN interacts with its environment in discrete time steps [**igel2003neuroevolution**]. 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 [**Miikkulainen:2010:ENN:1830761.1830902**].

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 [**RefWorks:31**].

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 [**RefWorks:31**]. This literature review compares direct and indirect encoding, which will be discussed in depth at a later stage in the Encoding Schemes.

2.10 NEAT

2.11 HyperNEAT

2.12 Fitness Functions

2.12.1 Objective Fitness

2.12.2 Novelty Fitness

2.12.3 Hybrid Fitness

2.13 Collective Construction Task

As has been outlined, the collective construction task requires that a team of agents cooperate by coordinating their behaviour in order to assemble various structures within their environment [14].

The general implementation of the collective construction task can be further separated into 3 distinct sub-tasks (for this outline, we consider the most basic task where a single robot in a team will be able to move and connect a building block to the structure):

1. First, the team of robots need to coordinate their efforts in order to be able to search through their immediate environment to find any resources that they could possibly use in the construction process. In our experiments, these resources were implemented as various types of 'building-blocks' that can only be connected to certain other types of blocks according to a predefined construction schema.

2. Second, once a robot has found a resource, it needs to take the resource to the designated 'construction site' so as to be able to connect it to the structure that is currently being built. We decided to use a designated construction site so as to prevent the robots from just starting new structures all over the place. These first 2 steps can be seen as an implementation of the standard aforementioned collective gathering task (can maybe add this as a footnote instead?) [14].

3. Third and finally, once the robot/s have brought the resources to the designated construction site/zone, it needs to be able to determine how and where to connect the resource to the structure. It will need to find an open side on the structure to which the block can be connected.

In order to perform this task successfully, the robot team will need to explore their environment and search for various resources that have been randomly scattered throughout the space in the most efficient and effective way possible.

'This project chose to investigate the collective construction task because it is sufficiently complex and it requires cooperative behaviour. It is also easy to manipulate this complexity as well as the levels of cooperation required as outlined in the previous section (in this case, this is outlined in Chapter1'

2.13.1 Collective Gathering Task

2.13.2 Existing Collective Construction Tasks

In this section, go on to provide some examples and research related to existing investigations of the collective construction tasks Can find some examples such as the ones with the templates that need to be filled and try to find any other examples

Chapter 3

Method

3.1 Evolutionary Algorithm: HyperNEAT

HyperNEAT [17] is an extension of NEAT (*Neuro-Evolution of Augmented Topologies*) [**StanleyMiikkulainen2002**], where ANNs are indirectly encoded using a CPPN (*Compositional Pattern Producing Network*) [**Stanley2007**]. HyperNEAT was selected as it has a number of benefits demonstrated in previous work [**DAmbrosio2013**], [**WatsonNitschke2015SSCI**]. This includes its capability to exploit geometric features such as symmetry, regularity and modularity in robot morphology and the task environment during controller evolution.

The nodes comprising each robot's ANN controller, connected by the CPPN, were placed in the substrate illustrated in figure **??**.

Each node in the substrate was placed at specific (x, y) locations in the two-dimensional geometric space of the substrate (x, y axes were in the range: [-1, 1]).

Connection weights in the controller were evolved via querying the CPPN for the weight of any connection between two points (x_1, y_1) and (x_2, y_2) by inputting (x_1, y_1, x_2, y_2) into the CPPN, which subsequently output the associated weight.

During HyperNEAT's evolutionary process, the CPPN was evolved via having nodes and connections added and removed, as well as connection weight values mutated [17].

Thus, the CPPN evolved a connectivity pattern across the geometry of the ANN via querying all the potential connections for their weights.

This connectivity pattern was effectively a function of the task and ANN geometry, which enabled HyperNEAT to exploit the structure (regularity, repetition and symmetry) of the task and robot morphology.

For example, there was symmetry in the robot morphology in terms of the positioning of sensors about each robot's periphery (figure **??**) and there was regularity and repetition in the collective construction task, in terms of repeating block types comprising modular and regular structures.

In the collective construction task, *modularity* was defined as the composition of modular structures (buildings in construction zones) from a sequence of connected blocks and *regularity* was defined as the same sequence of blocks repeated in a building.

Previous work has demonstrated that the indirect encoding of an evolved CPPN facilitates the evolution of robot controllers with increased task performance enabled by a compact representation of task and robot geometry [**DAmbrosioStanley2008**], [**WatsonNitschke2015SSCI**].

Table **??** presents the HyperNEAT parameters used in this study, where *delta* was angle between the (x_1, y_1, x_2, y_2) positions of nodes in the substrate. These parameter values were determined experimentally. All other HyperNEAT parameters not listed in table **??**, were set as in previous work [**DAmbrosioStanley2008**].

TABLE 3.1: Sensory configuration (number of sensors) for each morphology.

Morphology ID	Proximity Sensors	Ultrasonic Sensors	Color Ranged Sensors	Low-
1	5	3	1	
2	4	1	1	
3	0	0	1	
4	2	0	1	
5	2	2	1	

HyperNEAT was applied to evolve robot team (collective) behaviours, where teams were behaviourally and morphologically homogeneous teams meaning all robots in a given team used the same controller and sensory configuration.

3.2 ANN Controller

Each robot in the team used an ANN controller with N sensory input nodes, determined by the given morphology being evaluated (table **??**). Each robot's controller mapped sensory inputs, via a fully connected hidden layer, to two motor outputs, the robot's left and right wheels (figure **??**).

Figure **??** illustrates the sensory configuration for $N = 11$ (morphology 1), and the associated substrate and CPPN used by HyperNEAT.

For each robot morphology (table **??**), the sensors corresponding to the input layer of the controller was a circle N nodes distributed about a robot's periphery, where the exact geometric configuration corresponded to the morphology being evaluated (figure **??** illustrates morphology 1)[1].

The intermediate ANN hidden layer reflects the configuration of the input layer, preserving the geometry of the sensory input layer, that is, the direction of each sensor's FOV (figure **??**).

The ANN was initialized with random weights normalized to the range [-1.0, 1.0], with full connectivity between adjacent layers.

That being said, it was still possible to evolve partial connectivity via the CPPN generating a zero weight for a connection between 2 nodes from adjacent layers.

Collectively all sensors approximated up to a 360 degree *Field of View* (FOV).

3.2.1 Input Sensors

Each robot was equipped with various sensor types, where the exact sensor complement, including the relative position and direction on the robot depends upon the given experiment (section **??**) and morphology being evaluated (table **??**).

Each robot had N sensors corresponding to the N inputs comprising the robot's ANN sensory input layer (figure **??**), each with a range of r (portion of the environment's length).

A robot's sensory FOV was split into N sensor quadrants, where all sensors were constantly active for the duration of the robot's lifetime (unless stated otherwise in experiment conditions).

[1] Illustrations of all robot morphologies tested can be found at: `https://github.com/not-my-name/SSCI_Paper_Appendix`

The *nth* sensor returned a value in the normalized range [0.0, 1.0], in the corresponding *nth* sensor quadrant.

A value of 0.0 indicated that no blocks were detected and a value of 1.0 indicated that an object was detected at the closest possible distance to the given sensor.

Table **??** presents the different sensor types used in this study, where the functional properties of each sensor (range and FOV) were abstractions of corresponding physical sensors typically used on the Khepera III robots [**khepera3usermanual2013**].

In table **??**, range values are units defined in relation to the environment size (20 x 20) and FOV values are in radians.

Each morphology also included a special construction zone detection sensor that activated with a value in the range [0.0, 1.0] whenever a robot came into contact with a block that must be connected with other already connected blocks (existing construction zones).

The construction zone sensor calculated the squared Euclidean norm, bounded by a minimum observation distance, as an inversely proportional distance between *this* robot and the closest construction zone, where a value of 1.0 indicated the robot (pushing a block) was in contact with the construction zone and a value of 0.0 indicated that the robot (pushing a block) was the maximum possible distance from the closest construction zone.

Robots were unable to detect each other, thus all cooperative interactions were *stigmergic* [**BeckersHollandDeneubourg1994**] where robots interacted via pushing blocks into the environment's construction zone.

Furthermore, robots had no *a priori* knowledge of the construction schema, but rather must discover the construction schema rules by trial and error.

Also, once at least two blocks had been pushed and connected together this formed a construction zone (section **??**), that was then visible to each robot's construction zone sensor.

3.2.2 Movement Actuators

These movement actuators correspond to the output nodes for each ANN controller.

Two wheel motors control a robot's heading at constant speed. Movement is calculated in terms of real valued vectors (dx and dy). Wheel motors (L and R in figure **??**) need to be explicitly activated. A robot's heading is determined by normalizing and scaling its motor output values by the maximum distance a robot can traverse in one iteration (table **??**). That is:

$dx = d_{max}(o_1 - 0.5)$

$dy = d_{max}(o_2 - 0.5)$

Where, o_1 and o_2 are the motor output values, corresponding to the left and right wheels, respectively, producing an output in the range: [-1.0, 1.0]. These output values indicate how fast each respective wheel must turn. Equal output equates to straight forward motion and unequal output results in the robot rotating about its own axis. The d_{max} value indicates the maximum distance a robot can move in one simulation iteration (normalized to 1.0, table **??**).

3.3 Simulated Environment

These experiments were conducted in a simulated 2-dimensional environment that made use of the Redbridge simulator insert a reference

The robots that were used were simplified and abstracted versions of the Khepera robots (maybe find an online reference to what they are). This because they are

`"""TaskEnv"-eps-converted-to.pdf`

FIGURE 3.1: Example of the simulation environment. Robots search for randomly distributed type A, B, and C blocks (blue, green and red, respectively). Other colored and labeled blocks indicate those already connected in construction zones. Different coloured semi-circles emanating from each robot represent the field of view of currently active different sensor types (table **??**).

relatively simple and cheap to buy and they are also compatible with a wide range of sensors.

The physics of the simulated environment were relatively basic. It was implemented by using the JBox2D library for Java.

The environment through which the agents move and in which they interact is continous space but the environment results are evaluated using an underlying discretised grid. This is also used to 'snap' the connected resources to a fixed location on the grid once it has been attached to the construction zone. This is determined by checking the resource's distance to a connection surface as well as the its orientation relative to the target bloc. If it is within an acceptable range, the block automatically snaps into place and becomes fixed in space. ‘’ This toolkit is written in Java and it was chosen because of its consistent performance across a range of machines and its compatibility with various operating systems which would be useful in the testing phase as we had to run the simulations on a number of machines.

One of the reasons for choosing the MASON Java library is because it can easily be integrated with other Java-based tools, such as the Encog Machine Learning library, which is discussed earlier on in this section (**??**).

3.3.1 Construction Zone

A construction zone refers to an already existing structure that consists of two or more connected resources. Some examples of construction zones are shown in the following diagram (insert an illustration) and are indicated by the connected blocks changing to the same colour as well as having a label indicating the blocks type.

This is a collection of already constructed resources. It can be thought of as the current object that is currently under construction. This is the structure to which new resources should be connected in order to accomplish the construction of a single cohesive object rather than just a bunch of small structures that consist of 2 or 3 blocks.

Once a resource has been connected to a construction zone it becomes fixed in place. The entire construction zone is static and once a resource has been connected to it, it can not be disconnected again.

In order to encourage the construction of a single structure rather than just scattered structures that consist of 2 or 3 connected blocks, we restricted the number of construction zones to 3 per simulation. This means that 2 'free' resources can only be connected to each other thrice, the rest of the (subsequent) resources must be connected to one of those existing construction zones 'This only gets done for the first three pairs of blocks in order to encourage the development of a singular structure instead of several smaller ones'

The fact that the construction zones are only created by whichever 2 resources are connected first means that the location of the construction zone is different for each simulation run and depends on where the robots connect the first 2 blocks.

3.3.2 Resource Blocks

In order to represent the raw materials that are scattered throughout the environment for which the agents must search, we made use of basic squares with different physical properties.

These properties include: -¿ How many robots need to cooperate in order to be able to move the construction block -¿ Which types of blocks can be connected on which specific sides of the construction block (these are referred to as the underlying construction schema).

Three different types of construction blocks were implemented in order to define the complexity/difficulty of the task at hand

The complexity of the task is defined by the simulated environment and the resources that are present within it.

3.3.3 Construction Rules / Task Complexity

The level of required cooperation and the complexity of each experiment is determined by the types of blocks present in the environment (cooperation to move heavier blocks) and whether or not there are any construction rules present in the current schema.

An illustrated example of how the various difficulty levels are implemented is shown in figure **??**. Additionally, these configurations are outlined in more details later on in table **??**.

(Used to implicitly define the complexity of the simulated task at hand by controlling the simulated environment)

The connection schema specifies the number of each type of construction block that is to be placed in the environment for a specific experiment level.

Figure **??** provides an illustrated version of the various types and numbers of resource blocks that are present for each experiment level.

Level 1 was the least complex as it did not require any cooperation, given that in this case there were only type *A* blocks in the environment.

Level 2 was of medium complexity as there are equal numbers of type *A*, *B*, and *C* blocks in the environment, where block types *B* and *C* required at least two and three robots to push, respectively.

Level 3 was the most complex, as it required the same degree of cooperation as task level 2, though blocks had to be connected according to a construction schema. Figure **??** illustrates this construction schema, where the label on each of the four sides of each block type indicates what other block type can be connected to the given side.

`"""ConstructionSchema".PNG`

FIGURE 3.2: Task level 3 construction schema: A, B, and C are the block types. The label on each side of each block type indicates what block type can be connected to this side. An X label indicates that no block can be connected.

When block types have no letters on the sides it means that there were no blocks of that type present in the environment for that complexity level of the simulation (as with block types B and C in the first level of the illustrated schema)

3.4 Collective Construction Task

should this section not perhaps be part of the introduction section

The collective construction task was chosen as it is a task that benefits from autonomous robot groups that must exhibit robust collective behaviour in dynamic, noisy environments. Also, the collective construction task includes the notion of morphological damage to robots that may impede group task accomplishment.

The collective construction task requires that a team of agents cooperate by coordinating their behaviours in order to assemble various structures within their environment [reference] ¡verbatim from the honours paper¿ In this case, a team of homogeneous robots needs to search for various types of construction block 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 previously existing/created construction zone, then it needs to move the block towards another block in order to start a new construction zone. A new construction zone is created when any 2 free resources are successfully connected. Third: the robot must then try to connect the block it's carrying either to an existing construction zone or another block (to start a new construction zone). Once the robot has succcessfully connected the block to a construction zone, it releases the block and starts the process again by searching for another free 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 block types to each other.

3.4.1 Task Complexity

The complexity of the collective construction task is controlled by implementing the different connection schemas. By requiring the robots to cooperate in order to be able to move certain block types and implementing connection rules, it is necessary to evolve more complex controllers that are able to account for cooperative interactions which is a non-trivial task for most Evolutionary Algorithms (should maybe find a reference for this past paragraph? Currently its taken straight from page 5 of honours paper)

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.

Each of the different types of construction blocks are worth a different score depending on what level of cooperation they require (how many agents it takes to move the block) "I dont think there is any reward calculation for the complexity of the construction schema" The point is to simply deploy the agents with a more complex schema and then just see if they are able to solve the same task. I dont think its necessary to account for the difficulty level of the simulation in the fitness function calculation.

3.4.2 Construction Schema

The complexity/difficulty of a given simulation level is determined by the types of construction blocks present in the environment (as this determines the level of cooperation) as well as then underlying connection rules that govern which blocks can be connected to which other blocks and on their corresponding sides.

These underlying connection rules are specified using what will be referred to as a construction schema. This is essentially just a YAML file that indicates what block types can be attached to which side of each respective block. Whenever the robots attempt to connect 2 construction blocks, the simulator refers to this file and checks that the construction schema allows for these blocks to be connected on the indicated sides.

This is almost like an abstract representation of how receptors and hormones work in the brain, i think. Or is it something to do with DNA replication? Eitrher way, its like the lock-key model for proteins

This is analogous to the way in which puzzle pieces only fit together on specific sides and orientations.

When a robot attempts to attach 2 blocks to each other, the connection rules for both blocks must be satisfied in order for a successful connection to be made.

3.5 Team Controller

'This is from the GECCO Penultimate paper pg2, remember to go get the references and the diagrams'

Each robot's ANN controller comprised N sensory input and hidden nodes, connected hidden layer to two motor outputs (controlling the robot's left and right wheels). Nodes were arranged as a substrate where the number of input and hidden nodes was determined by a give robot morphology. Each substrate node was placed at specific (x,y) locations in the substrate's two-dimensional geometric space. Sensor nodes of the substrate approximated up to a 360deg sensory FOV, where the FOV was dependent on the morphology used.

Chapter 4

Experiments

SSCI -¿ HyperNEAT objective across 5 morphologies and 3 complexity levels

4.1 Experiment Outline

Experiments were used to evaluate the *morphological robustness* of ANN controllers evolved using the HyperNEAT algorithm for robot teams that must accomplish a collective construction task of increasing complexity (section **??**).

We measured the average comparative task performance of controllers evolved for a given team morphology and task complexity (this is calculated during the 'Baseline' step) where such controllers were then transferred to and re-evaluated across other team morphologies.

Thus, teams that achieved an average task performance that was not significantly lower across all *re-evaluated* morphologies were considered to be *morphologically robust*.

This study comprised five experiment sets, where the first four experiment sets evolved controllers given team morphologies $1 - 4$ (table **??**) for three levels of increasing task complexity (table **??**).

The fifth experiment set investigated the *co-adaptation* of team morphology and behavior, where morphology 5 (table **??**) was used as the initial sensory configuration for all robots in the team.

This fifth experiment set was included in order to gauge if co-adapting behavior and morphology yielded any benefits in this collective construction task as it did in related collective behavior tasks [**HewlandNitschke2015**].

Each experiment set [1] tested a team of 15 robots in a bounded (20 x 20 units), two-dimensional, and continuous simulated environment along with a collection of randomly distributed resource blocks of type A, B, and C (table **??**tab:simParameters).

During initialization of an experiment run, the robots and the resource blocks were placed at random locations and with random orientations.

A construction schema (table **??**) dictated the sequence of block types that must be connected together in order that a specific structure be built [14].

Figure **??** presents an example of the team of 15 robots working to solve the collective construction task in the simulation environment containing a distribution of five of each block type (A, B and C), colored blue, green and red, respectively. Other colored blocks in the environment indicate those already connected in construction zones (three illustrated).

The purple, blue and green semi-circles emanating from each robot represent the FOV of active sensors, where the different colors correspond to different sensor types (table **??**).

[1]Source code for all experiments is online at: `https://github.com/not-my-name/ExperimentsRerun`

As the purpose this study was to demonstrate the morphological robustness of HyperNEAT evolved controllers for a collective behavior task of increasing complexity, the first two versions of the collective construction task required no cooperation and some degree of cooperation, respectively, though any block could be connected to any other block. Where as, the most complex version of the task required cooperation and block types to be connected according to a construction schema (table **??**)

The fittest controller evolved for each experiment set (yielding the highest absolute task performance) was then *evaluated* for morphological robustness across all other morphologies.

For example, the fittest controller evolved for morphology 1 was evaluated across morphologies $2 - 4$ and the average task performance calculated across all evaluation runs.

Each set of experiments (testing conditions?) can be broken into the following distinct stages: 1] Evolution 2] Baseline Calculation 3] Evaluation

4.1.1 Evolution Phase

For controller evolution, each experiment applied HyperNEAT to evolve team behavior for 15 robots over 100 generations, where a generation is comprised of 5 *team lifetimes* (with 1000 simulation iterations per lifetime).

A single evolution phase can broken down into the following distinct steps: 1] Simulation - The team of agents are deployed into the simulated environment along with the necessary resource blocks. The agents must then operate within this environment for a 'lifetime', attempting to solve the task at hand. A single 'Simulation' step consists of 5 lifetimes, 2] Fitness Calculation - At the end of each lifetime, the environment is analysed and compared with the desired output. At the end of a 'Simulation' step, the fitness of the controller is calculated using an average of these values and the provided fitness function. This value indicates a controller's suitability to solving the collective construction task. 3] HyperNEAT Evolution - The fitness scores obtained in the previous steps are passed to the Evolutionary Algorithm (along with their respective controllers) at which point the subsequent generation of controllers are created

Each team lifetime tested different robot starting positions, orientations, and block locations in the simulation environment.

4.1.2 Baseline

The purpose of this stage is to simply calculate an accurate average fitness score for each optimal controller. This average fitness provides a more accurate indication of a controller's suitability to solving the task.

During this simulation stage, the parameters are the same as for the evolution phase in which the controller was evolved (ie. the same length of lifetime (simulation steps), same block configuration, same construction schema).

In order to obtain this 'baseline' average score, the absolute fittest controller for each experimental condition is redeployed in the simulator and allowed to try and solve the task.

A single 'baseline' attempt in the simulator is equivalent to 1 team lifetime and this is repeated for 20 lifetimes and an average fitness score is calculated.

During this stage, no Machine Learning processes or evolutionary steps are performed.

4.1.3 Evaluation

Each evaluation run was *non-evolutionary*, where controllers were not further evolved, and each evaluation run in the simulator was equivalent to one team lifetime.

Evaluation runs were repeated 20 times for a given morphology, in order to account for random variations in robot and block starting positions and orientations.

For each fittest controller, evaluated in a given morphology, an average baseline task performance was calculated over these 20 runs, and then an overall average task performance was computed across all evaluated morphologies, producing an average fitness score that indicated a particular controller's ability to adapt to changes in its sensory configuration (morphology).

As per this study's objectives, these morphological re-evaluation runs tested how robust the fittest evolved controllers (for a given morphology) were to variations in that morphology. Thus, re-evaluating the fittest controllers on other morphologies emulated sensor loss due to damage or new robot morphologies introduced due to changing task constraints.

The above table shows the overall outline for the resources used across the various experiments. For the experiments that did not make use of all of the resources listed above, they simply used a subset of the above resources.

Table 4.1: Experiment, Neuro-evolution and Sensor Parameters

Generations	100	
Sensors per robot	11, 8, 4, 6, random	
Evaluations per genotype	5	
Experiment runs	20	
Environment length, width	20	
Max Distance (Robot movement per iteration)	1.0	
Team size	15	
Team Lifetime (Simulation iterations)	1000	
Lifetimes per generation	5	
Type A blocks (1 robot to push)	15	
Type B blocks (2 robots to push)	15	
Type C blocks (3 robots to push)	15	
4*Mutation rate	Add neuron	0.25
	Add connection	0.008
	Remove connection	0.002
	Weight	0.1
Population size	150	
Survival rate	0.3	
Crossover proportion	0.5	
Elitism proportion	0.1	
CPPN topology	Feed-forward	
CPPN inputs	Position, delta, angle	
Sensor	Range	FOV
Proximity Sensor	1.0	0.2
Ultrasonic Sensor	4.0	1.2
Ranged Colour Sensor	3.0	1.5
Low-Res Camera	3.0	1.5
Colour Proximity Sensor	3.0	3.0

TABLE 4.2: Task Complexity. Note: Task level 3 includes a construction schema (figure **??**).

Construction Task Complexity	Level 1	Level 2	Level 3
Type A blocks (1 robot to push)	15	5	5
Type B blocks (2 robots to push)	0	5	5
Type C blocks (3 robots to push)	0	5	5
Construction schema	No	No	Yes

Chapter 5

Results & Discussion

Results indicate that, for both novelty and objective search, given increasing task complexity, there was no significant difference in average task performance between the controllers evolved in any morphology and then evaluated in other morphologies.

However, there were two exceptions to this result. First, novelty search applied in task complexity level 1 to evolve controllers evaluated in morphology 3 yielded significantly lower average task performance.

Second, objective search applied in task complexity level 2 to evolve controllers in morphology 1, where similarly, the fittest controller evaluated in morphology 3 yielded a significantly lower task performance.

In these cases, one may observe that there is a large difference in the number of sensors used by morphology 1 versus morphology 3 (table from AAMAS Abstract), indicating that controllers evolved for the high sensory complement (and thus functionality) of morphology 1 are not readily transferable to a simpler sensory configuration (with much less functionality).

However, this was not the case for novelty search applied in task level 2 or objective-search applied in task level 1.

The key results were thus two-fold. First (supporting the first hypothesis), team behaviours evolved by novelty search, for ALL morphologies and task complexity levels, significantly out-performed team behaviours evolved with objective search for corresponding morphologies and task complexity levels. This result is supported by the following previous work [**DidiNitschke2016SSCI**], [**DidiNitschke2016**], [8], [9], [10]

Second (refuting the second hypothesis), results indicated that, for all team morphologies and levels of task complexity evaluated, both novelty and objective search were effective in evolving morphologically robust controllers.

That is, for any given morphology, there was no significant difference in average task performance between the fittest novelty search evolved controllers and "re-evaluation" of these controllers in the other morphologies.

The same result was observed for objective search. This second result indicates that while novelty search yields advantages over objective search in terms of evolving high task performance team behaviours, it yields no benefits over objective search for evolving morphologically robust controllers in the collective construction task for the given team morphologies. However, the suitability of novelty search for evolving morphologically robust controllers in complex collective behaviour tasks is the topic of ongoing research.

To address the research objectives, we compared the average task performance results of team behaviour evolution directed by objective versus novelty search. For each experiment, task performance was the number of blocks connected in construction zones over a team's lifetime, where the maximum task performance was taken at

"""Baseline_Level_1".png

"""Average_Eval_Level_1".png

FIGURE 5.1: *Left column:* Baseline task performance for evolved controllers (*task level 1*) given morphologies $1 - 5$ (depicted from left to right). *Right column:* Average task performance given the fittest controller evolved for each respective morphology ($1 - 5$, shown left to right) evaluated across all other morphologies. For example: Left-most plot is average task performance of fittest controller evolved for morphology 1, evaluated across morphologies $2 - 5$. Right-most plot is the average task performance of fittest controller evolved for morphology 5, evaluated across morphologies $1 - 4$.

each run's end and an average task performance calculated over 20 runs (the "Evaluation Phase").

For each evolution phase of an experiment set, the average maximum task performance of controllers evolved for a given morphology and level of task complexity, was recorded.

Specifically, this task performance was calculated by running the absolute fittest controller evolved after 20 evolutionary runs (for a given a morphology and task level), in the same morphology over 20 non-evolutionary runs (baseline calculations in **??**).

This is previously referred to as the baseline average score for a specific controller (section **??**).

This is presented in figures **??**, **??** and **??** (left column), where controllers were evolved given morphologies 1-5 (table **??**). In figures **??**, **??** and **??** (left column), these results are presented from left to right.

For example, average task performance results for morphology 1 are plotted on the left-most side and average task performance results for morphology 5 are plotted on the right-most side.

For each evaluation phase of an experiment set, the fittest controller evolved for a given morphology and task complexity level was evaluated across all other morphologies (for the same level of task complexity), and an average task performance computed over 20 runs.

These morphological evaluation results are presented in figures **??**, **??** and **??** (right column).

Each of the five plots (from left to right) in each figure corresponds to the fittest controller evolved for each of the five morphologies and re-evaluated in all other morphologies. For example, the left-most plot presents the average task performance

"""Baseline_Level_2".png

"""Average_Eval_Level_2".png

FIGURE 5.2: *Left column:* Baseline task performance for evolved controllers (*task level 1*) given morphologies $1 - 5$ (depicted from left to right). *Right column:* Average task performance given the fittest controller evolved for each respective morphology ($1-5$, shown left to right) evaluated across all other morphologies. For example: Left-most plot is average task performance of fittest controller evolved for morphology 1, evaluated across morphologies $2-5$. Right-most plot is the average task performance of fittest controller evolved for morphology 5, evaluated across morphologies $1 - 4$.

of the fittest controller evolved in morphology 1 and re-evaluated on morphologies $2 - 5$.

Where as, the right-most plot presents the average task performance of the fittest controller evolved given morphology 5 (as the initial sensory configuration) and re-evaluated on morphologies $1 - 4$.

To gauge the impact of a given team morphology (table **??**) in company with a given level of task complexity (table **??**), the *t-test* [**FlanneryTeukolsky1986**] ($p < 0.05$), was applied in pair-wise comparisons between sets of controller evolution results[1] (figures **??**, **??** and **??**, left column).

Within each given level of task complexity, no statistically significant difference was found between controllers evolved given morphologies $1 - 5$ (controller evolution experiments $1 - 5$).

Controller evolution experiments $1 - 4$ were those implementing controller evolution in fixed sensory configurations (morphologies $1 - 4$). Where as, controller evolution experiment 5 used morphology 5 as the initial sensory configuration and subsequently co-adapted behavior (controller) and morphology (complement of sensors).

The lack of statistical difference between controllers evolved given morphologies $1 - 4$ (table **??**) indicates that these sensory configurations were not sufficiently different so as to result in significantly different average maximum task performances.

Also, the lack of any significant difference between the average maximum task performance of controllers evolved given morphologies $1-4$ and behavior-morphology co-adaptation (starting with morphology 5), supports previous results demonstrating

[1] Statistical test results for pair-wise comparisons for the fittest evolved controllers (for a given morphology) tested in each other morphology (individually) is online at: `https://github.com/not-my-name/SSCI_Paper_Appendix`

`"""Baseline_Level_3".png`

`"""Average_Eval_Level_3".png`

FIGURE 5.3: *Left column:* Baseline task performance for evolved controllers (*task level 1*) given morphologies $1 - 5$ (depicted from left to right). *Right column:* Average task performance given the fittest controller evolved for each respective morphology ($1 - 5$, shown left to right) evaluated across all other morphologies. For example: Left-most plot is average task performance of fittest controller evolved for morphology 1, evaluated across morphologies $2 - 5$. Right-most plot is the average task performance of fittest controller evolved for morphology 5, evaluated across morphologies $1 - 4$.

that behavior-morphology co-adaptation yields at least comparable task performance benefits (compared to fixed morphology controller evolution) in collective behavior tasks [**HewlandNitschke2015**].

However, to address this study's main objective it was necessary to ascertain the morphological robustness of the fittest controller evolved in each morphology when re-evaluated in all other morphologies.

To gauge the morphological robustness of the fittest controllers evolved for a given morphology ($1 - 5$), and a given task complexity, we applied the t-test in pair-wise comparisons of two result data sets.

First, the average maximum task performances yielded by controller evolution in morphologies $1 - 5$ and second, the average maximum task performances yielded from evaluating the fittest controller evolved for a given morphology in all other morphologies (section **??**).

Statistical test results indicated no significant difference (with one exception) between average task performance results yielded by baseline calculations and morphological evaluation experiments for all task complexity levels.

Specifically, the average maximum task performance yielded by controllers evolved given morphologies 2-5 (figures **??**, **??**, **??**, left column) was not significantly lower than the average task performance yielded by the fittest controllers (evolved in morphologies $1 - 5$), and then re-evaluated on other morphologies (figures **??**, **??**, **??**, right column).

The exception was morphology 1 in task complexity levels 2 and 3.

In these tasks, the fittest controller evolved for morphology 1, yielded a significantly higher average baseline task performance than that yielded when this fittest controller was evaluated in morphologies $2 - 5$.

Hence, these results indicate that controllers evolved by HyperNEAT for a given morphology (table **??**), overall have the capacity to continue to effectively operate when transferred to other morphologies.

This result was found to hold for all four of the five morphologies that controllers were evolved for, and for all levels of task complexity tested (table **??**).

The efficacy of HyperNEAT for evolving morphologically robust controllers is further supported by the controller evolution experiments that used morphology 5 (section **??**).

In this case, the number of sensors was adapted meaning that team behavior and morphology were co-adapted.

Specifically, these controller evolution experiments began with the sensory configuration of morphology 5 (table **??**) and enabled and disabled sensor connections to better couple morphology with the evolved controller.

Hence, the fittest controller evolved in this case often corresponded to a sensory configuration dissimilar to morphology 5 (the initial sensory configuration).

Results indicated that the fittest controller evolved for morphology 5, when evaluated across other morphologies, yielded an average task performance that was statistically comparable to the average baseline task performance yielded given morphology 5.

This result was observed for all three task complexity levels (figures **??**, **??**, and **??**, right column).

Thus, controllers evolved for fixed morphologies ($1-4$, table **??**), were found to be *morphologically robust*, as there was no significant difference in average maximum task performance when the fittest controller (evolved for morphology $1-4$), was evaluated using other morphologies.

Furthermore, the fittest controller evolved for an adaptive morphology (5, table **??**), was similarly found to be morphologically robust, given that the average maximum task performance yielded when this fittest controller was evaluated in other morphologies ($1-4$), was comparable to the average maximum task performance yielded from morphology 5 controller evolution experiment (baseline average fitness).

This is theorized to be a result of the complexity of co-adapting effective controller-morphology couplings [**PfeiferBongard2006**] within limited periods of artificial evolution (100 generations in these experiments, table **??**), offset by the transference of evolved *connectivity patterns* [**GauciStanley2010**] as functional controllers across varying robot morphologies [**RisiStanley2013**].

Such connectivity patterns encode behaviors that do not rely upon specific sensory-motor mappings in controllers and thus do not necessitate specific task environment configurations, such as specific numbers of agents or objects.

This in turn facilitates the transfer of controllers across varying team morphologies [**verbancsics˙evolving˙2010**], [**DidiNitschke2016SSCI**], [**DidiNitschke2016**].

These results are corroborated by related work [**RisiStanley2013**], [**WatsonNitschke2015SSCI**], and contribute further empirical evidence that HyperNEAT yields significant benefits in evolving robot controllers that effectively operate in other morphologies.

That is, this study further demonstrated HyperNEAT's capability to exploit geometric properties such as regularity, repetition and symmetry in robot morphology and environment [17], where such modularity and geometric properties are encoded in evolved connectivity patterns.

This is prevalent in this study, as the configuration of sensors on each robot's periphery was symmetrical for all morphologies tested (section **??**). Not sure if the above is correct since Morph 5 was a random distribution

Also, the collective construction task required that blocks be connected together in a repeated manner in a symmetrical bounded simulation environment (20x20 units, table **??**).

The capability of HyperNEAT evolved controllers to operate in different morphologies is further supported by other research [**verbancsics˙evolving˙2010**] demonstrating that evolved indirect sensory-motor mappings can encapsulate effective behaviors with relatively few task environment and robot geometric relationships, such as desired positions and angles between robots and different object types.

The efficacy of HyperNEAT for evolving morphologically robust controllers for collective behavior tasks of varying complexity is also supported by related research in *multi-agent policy transfer* [**verbancsics˙evolving˙2010**], [**DidiNitschke2016SSCI**], [**DidiNitschke2016**].

Policy transfer methods facilitate the transfer of behaviors across tasks of increasing complexity or between dissimilar tasks.

Such studies have demonstrated that HyperNEAT is an effective method for evolving behaviors in one collective behavior task and then transferring the evolved behavior to a related but more complex task (for example, where robots have more complex sensory-motor configurations to process increased task complexity) with relatively little loss in average team task performance.

However, we hypothesize that the morphological robustness of HyperNEAT evolved controllers demonstrated across all morphologies tested (table **??**) and all levels of task complexity (table **??**) was facilitated by the use of morphologically and behaviorally homogenous teams.

Specifically, one controller was evolved for all robots in a team and all robots used the same sensory configuration, meaning all robots had the same *collective behavior geometry* [**DAmbrosioLehmanStanley2010**]. This in turn simplified the transfer of evolved controllers across varying morphologies with no significant degradation in average task performance.

Hence, overall, this study's results demonstrate that HyperNEAT is an appropriate method for evolving morphologically robust controllers.

That is, controllers that are fully functional in a range of team morphologies.

In order to ascertain how well HyperNEAT evolved controllers generalize, ongoing research is evaluating the evolution of morphologically robust behaviors in behaviorally and morphological heterogenous teams for complex collective behavior tasks that are irregular and without repetition or symmetry.

Furthermore, current research is comparing HyperNEAT to related evolutionary approaches that have demonstrated controllers able to accomplish multiple disparate tasks in dynamic environments [**IzquierdoTorres2008**], as well as direct encoding neuro-evolution methods such as NEAT [**StanleyMiikkulainen2002**].

Chapter 6

Conclusions

This research presented a study on the efficacy of HyperNEAT for evolving *morphologically robust* behaviors for homogenous robot teams that must solve a collective behavior task of increasing complexity. That is, the average maximum task performance of behaviors evolved for a given team morphology (robot sensory configuration) that was then transferred to a different team morphology. Controllers that did not yield degraded task performance when transferred to another morphology were considered to be morphologically robust. The objective was to test and evaluate methods that generate morphological robust behaviors, where varying morphologies emulated sensor damage or intentional changes to the sensory systems of robotic teams.

Results indicated that HyperNEAT was appropriate for generating morphologically robust controllers for a collective construction task of increasing complexity. This task required robots to cooperatively push blocks such that they connected together to form structures. Task complexity was regulated by the number of robots required to push blocks and a construction schema mandating that specific block types be connected in specific ways. These results support the notion that developmental neuro-evolution methods, such as HyperNEAT, are appropriate for controller evolution in robotics applications where robot teams must adapt during their lifetime to damage or otherwise must dynamically adapt their sensory configuration to solve new unforseen tasks.

Appendix A

Frequently Asked Questions

A.1 How do I change the colors of links?

The color of links can be changed to your liking using:
`\hypersetup{urlcolor=red}`, or
`\hypersetup{citecolor=green}`, or
`\hypersetup{allcolor=blue}`.
If you want to completely hide the links, you can use:
`\hypersetup{allcolors=.}`, or even better:
`\hypersetup{hidelinks}`.
If you want to have obvious links in the PDF but not the printed text, use:
`\hypersetup{colorlinks=false}`.

Bibliography

[1] G. Beni. “From Swarm Intelligence to Swarm Robotics”. In: *Proceedings of the First International Workshop on Swarm Robotics.* Santa Monica, USA: Springer, 2004, 1–9.

[2] J. Bongard, V. Zykov, and H. Lipson. “Resilient machines through continuous self-modeling”. In: *Science* 314(5802) (2006), pp. 1118–1121.

[3] Luiz Chaimowicz et al. “An architecture for tightly coupled multi-robot cooperation”. In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164).* Vol. 3. IEEE. 2001, pp. 2992–2997.

[4] A. Cully et al. “Robots that can adapt like animals”. In: *Nature* 521(1) (2015), pp. 503–507.

[5] Agoston E Eiben and James E Smith. “Introduction to evolutionary computing”. In: 53 (2003).

[6] William G Fenton, T. Martin McGinnity, and Liam P Maguire. “Fault diagnosis of electronic systems using intelligent techniques: A review”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 31.3 (2001), pp. 269–281.

[7] D. Floreano, P. Dürr, and C. Mattiussi. “Neuroevolution: from architectures to learning”. In: *Evolutionary Intelligence* 1.1 (2008), pp. 47–62.

[8] Jorge Gomes and Anders L Christensen. “Generic behaviour similarity measures for evolutionary swarm robotics”. In: *Proceedings of the 15th annual conference on Genetic and evolutionary computation.* ACM. 2013, pp. 199–206.

[9] Jorge Gomes, Pedro Mariano, and Anders Lyhne Christensen. “Devising effective novelty search algorithms: A comprehensive empirical study”. In: *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference.* ACM, 2015, pp. 943–950.

[10] Jorge Gomes, Paulo Urbano, and Anders Lyhne Christensen. “Evolution of swarm robotics systems with novelty search”. In: *Swarm Intelligence* 7.2-3 (2013), pp. 115–144.

[11] Faustino John Gomez and Risto Miikkulainen. “Robust non-linear control through neuroevolution”. In: (2003).

[12] R. Kube and H. Zhang. “Collective robotics: from social insects to robots”. In: *Adaptive Behaviour* 2.2 (1994), pp. 189–218.

[13] Joel Lehman and Kenneth O Stanley. “Abandoning objectives: Evolution through the search for novelty alone”. In: *Evolutionary computation* 19.2 (2011), pp. 189–223.

[14] G. Nitschke, M. Schut, and A. Eiben. “Evolving Behavioral Specialization in Robot Teams to Solve a Collective Construction Task”. In: *Swarm and Evolutionary Computation* 2.1 (2012), 25–38.

[15] W. Shen, P. Will, and B. Khoshnevis. "Self-Assembly in Space via Self-Reconfigurable Robots". In: *IEEE International Conference on Robotics and Automation.* Taipei, Taiwan: IEEE Press, 2003, pp. 2516–2522.

[16] K. Stanley. *Efficient Evolution of Neural Networks Through Complexification. Ph. D. Dissertation.* Austin, USA: Department of Computer Sciences, The University of Texas, 2004.

[17] K. Stanley, D'Ambrosio, and J. Gauci. "Hypercube-based indirect encoding for evolving large-scale neural networks". In: *Artificial Life* 15.1 (2009), pp. 185–212.

[18] Vandi Verma et al. "Real-time fault diagnosis [robot fault diagnosis]". In: *IEEE Robotics & Automation Magazine* 11.2 (2004), pp. 56–66.

[19] Justin Werfel et al. "Distributed construction by mobile robots with enhanced building blocks". In: *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on.* IEEE, 2006, pp. 2787–2794.

[20] X. Yao. "Evolving artificial neural networks". In: *Proceedings of the IEEE* 87.9 (1999), pp. 1423–1447.