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

December 14, 2018

Declaration of Authorship

I, A. Ruben Putter, declare that this thesis titled, "Evolving Morphological Robustness for Collective Robotics" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"*Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism.*"

Dave Barry

Abstract

A. Ruben Putter

Evolving Morphological Robustness for Collective Robotics

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor...

Contents

List of Figures

List of Tables

List of Abbreviations

LAH	**L**ist **A**bbreviations **H**ere
WSF	**W**hat (it) **S**tands **F**or

Physical Constants

Speed of Light $\quad c_0 = 2.997\,924\,58 \times 10^8\,\mathrm{m\,s^{-1}}$ (exact)

List of Symbols

a	distance	m
P	power	W ($\mathrm{J\,s^{-1}}$)
ω	angular frequency	rad

For/Dedicated to/To my...

Chapter 1

Introduction

1.1 Introduction

From the humble origins of the single-celled organism, to the great blue whale and the naked mole rat. The evidence as to the problem solving power of mother nature is boundless. In every nook and cranny on the planet we can find some specialised organism that has adapted in such a way as to be the most successful creature to survive in that habitat. Over hundreds and thousands of generations, these creatures have undergone changes on the genetic level, discovering which characteristics are useful for survival and which ones are suboptimal, resulting in the organisms themselves failing to pass on their genes

now just write as if you dont need references and then go back and check which lines you can add references

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 [11], making it unsurprising that computer scientists have resorted to researching natural processes for inspiration.

'There is no a priori reason why machine learning must borrow from nature. A field could exist, complete with well-defined algorithms, data structures, and theories of learning, without once referring to organisms, cognitive or genetic structures, and psychological or evolutionary theories. Yet at the end of the day, with the position papers written, the computers plugged in, and the programs debugged, a learning edifice devoid of natural metaphor would lack something. It would ignore the fact that all these creations have become possible only after three billion years of evolution on this planet. It would miss the point that the very ideas of adaptation and learning are concepts invented by the most recent representatives of the species Homo sapens from the careful observation of themselves and life around them. It would miss the point that natural examples of learning and adaptation are treasure troves of robust procedures and structures. Fortunately, the field of machine learning does rely on natures bounty for both inspiration and mechanism. Many machine learning systems now borrow heavily from current thinking in cognitive science, and rekindled interest in neural networks and connectionism is evidence of serious mechanistic and philosophical currents running through the field. Another are where natural example ahs been trapped is in work on genetic algorithms and genetics-based machine learning. Rooted in the early cybernetics movement, progress has been made in both theory and application to the point where genetics-based systems are finding their way into everyday commercial use' [14].

outline the 2 most powerful problem solving tools found in nature: brain and evolution Assuming that the human brain is the greatest problem solving tool in nature, then by extension we can argue that the process of evolution is the greatest problem solver since it is the process that discovered the brain. Which is the best problem

solver, the brain which discovered the wheel and all of the technological advances we can see today, or the process that produced the brain.

'Of the two natural archetypes of learning available to us - the brain and evolution - why have genetic algorithm researchers knowingly adopted the "wrong" metaphor?' [14] find some more information on why this is the 'wrong' metaphor? also, find some more information explaining that these are the 2 archetypes for learning 'One reason is expedience. The processes of natural evolution and natural genetics have been illuminated by a century of enormous progress in biology and molecular biology. in contrast, the brain, though yielding some of its secrets, remains largely and opaque gray box; we can only guess at many of the fundamental mechanisms contained therein.' [14].

Automated production of controllers for single agents attempting to accomplish increasingly complex tasks One of the central themes of research in mathematics and computer science is the design and development of automated problem solvers [11].

The current trend in increasing levels of computerisation coupled with the increasing complexity of tasks that need to be performed has led to a growing demand for the automated production of these problem solvers [11].

Given the levels of complexity and non-linearity of real-world problems, traditional methods of controller design have become impractical since they mostly rely on linear mathematical models [18].

additional complexity inherent in producing organised collective behaviour

An open problem in *collective* [26] and *swarm robotics* [3] is ascertaining appropriate sensory-motor configurations (morphologies) for robots comprising teams that must work collectively given automatically generated controllers [12].

The complex nature of the intricate dynamics inherent in producing self-organised behaviour makes it practically impossible to design these multi-robot team controllers by hand [15]. can maybe reword this somehow? seems a bit lengthy and out of place

"Evolutionary Computing is a branch of computer science research that draws its inspiration from the Darwinian process of evolution through natural selection [10]."

mention here the different biological inspirations that were considered. The brain and the process that found the brain. Present some information on how the both of them work

In this particular case, researchers have looked into using the most powerful problem solving tool that has ever existed, evolution through the process of natural selection. Starting from single-celled organisms, the process of evolution has produced the incredibly diverse range of organisms that we can see on Earth today.

information on finding the best way to find the best body brain coupling In *evolutionary robotics* [9], a popular method is to co-evolve robot behaviors and morphologies [28], [29], [4], [2].

motivations for different encodings

Within the purview of such *body-brain* co-evolution, indirect encoding (developmental) methods have been effectively demonstrated for various single-robot tasks [30], [21], [5].

In cases such as the above, the behaviour and the physical structure of the agent are both evolved at the same time. During the evolution processes (parent selection, mutations/cross-overs, reproduction) affect both of these criteria. In this case, both the behaviour (structure of the ANN, where behaviour is the action produced based on a certain input) and the physical structure/morphology (the actual configuration of the agent's sensory input) are both encoded in the genotype and these properties are affected by the evolutionary operators. above explanation is in my own words, but should maybe find some references in order to back this up

'Of course, simple expedience is not the best reason for adopting a particular course of action, and at first glance, it is not at all obvious why learning in natural or artificial minds should be anything like the adaptation that has occurred in evolution. Yet there is an appealing symmetry in the notion that the mechanisms of natural learning may resemble the processes that created the species possessing those learning processes. Furthermore, the idea that the mind is subject to the same competitive-cooperative pressures as evolutionary systems has achieved some currency outside of GA circles ' [14].

'Despite these suggestions, genetic algorithms and genetics-based machine learning have often been attacked on the grounds that natural evolution is simply too slow to accomplish anything useful in an artificial learning system; three billion years is longer than most people care to wait for a solution to a problem. However, this slowness argument ignores the obvious differences in timescale between natural systems and artificial systems.' [14].

ARGUMENT FOR COMPLEXITY ACHIEVED USING EVOLUTIONARY PROCESSES ' A more fundamental fault is that this argument ignores the robust complexity that evolution has achieved in its three billion years of operation. The "genetic programs" of even the simplest living organisms are more complex than the most intricate human designs' [14]

should maybe find the original references mentioned below 'Waddington (1967) presents more sophisticated probabilistic arguments that actual evolutionary processes have achieved complexity in existing species that is incommensurate with an evolutionary process using only selection and mutation. Although such argmuments were originally meant to challenge evolutionary theory, genetic agorithmists see no such challenge. Insted, the high speed-to-complexity level observed in nature lends support to the notion that reproduction, recombination, and the processing of building blocks result in the rapd development of appropriate complexity. Moreover, this speed is not purchased at the cost of generality. The mechanisms of genetics and genetic algorithms permit relative efficiency across a broad range of problems' [14]. Although the process of evolution may take hundreds and/or thousands of generations to find an appropriate solution, it is able to produce a level complexification that exceeds that of the solutions found by neurocomputation (the brain process).

given the added complexity inherent in this body-brain coonfiguration, we chose to focus this research solely on evolving the brain and instead using a preconfigured set of input sensors and then comparing the resultant task performance. In this study, we implement increasing complexity in the task that needs to be solved in order to test the robustness of the brains that have been evolved with respect to unexpected changes in their input. We are not investigating the algorithms ability to find the best input configuration body-brain coupling, instead we are trying to determine how robust the evolved controller/brain is with respect to unexpected and undesirable changes in its sensory input configuration. We would like to see how the solution of the problem has been 'embedded' in the network/controller itself. If the solution to the task contained in the network is dependent on the type of input that it can receive.

By transferring a controller across several input configuration (morphologies), we can determine if the controller even needs to know what type of input it is receiving. When being transferred to another input, the controller does not know whether it is receiving input from the same type of sensor as they were evolved with. For example, the controller was evolved with an Infra-red sensor at input node 1, but during one of the evaluation trials where it is transferred to another input config, the input at node 1 might be coming from a Camera or an Ultrasonic sensor.

This might be going into too much detail considering how basic the simulator was and how little detail we could really control.

1.1.1 Collective Construction Task

–the below can be part of research contributions?? explaining some related research– However, with few exceptions [1], [44], [20] there has been relatively little work that evaluates developmental methods to evolve behavior and morphology for collective robotics tasks, excluding related research in self-assembling and reconfigurable collective robotics [37], [38].

Specifically, this study focuses on developmental methods to evolve controllers that exhibit consistently robust behaviors such that robot teams effectively adapt to the loss of sensors or new sensory configurations without significant degradation of collective task performance, that is, *morphologically robust behavior*.

HyperNEAT was selected as it is a developmental encoding NE method with demonstrated benefits that include exploiting task geometry to evolve modular and regular ANN controllers with increased problem-solving capacity [42], [7].

The *Collective construction* task [46] was selected since it benefits from fully automated robot teams that must exhibit robust behavior to handle changing task constraints, potential robot damage and sensor noise. For example, collective construction of functional structures and habitats in remote or hazardous environments [45].

As in related work [46], the task was for robots to search the environment for *building-block* resources, then move them such that are connected to other blocks. The task is solved if the team connects all blocks forming a structure during its *lifetime*, and this is equated with optimal task performance. However, the task is considered partially solved if only a sub-set of the blocks are connected by the team, though in such instances, team task performance is proportional to the number of blocks connected. In this study, task complexity was equated with the number of robots needed to collectively push blocks together (cooperation) to be connected as a structure and whether the blocks must be connected in a specific sequence, that is, according to a *construction schema*, table **??**.

Hence, we report a preliminary investigation into developmental methods (HyperNEAT is tested in this case study) for evolving ANN controllers that are robust to morphological change in robotic teams or swarms that must operate in dynamic, noisy and hazardous environments.

[There is probably something that can also be said about how this applies to finding solution controllers that are highly effective in extremely specific tasks, but barely effective at all when it comes to attempting any other task]

Although Reinforcement Learning (RL) techniques can theoretically solve a number of problems without examples of correct behaviour, in practice they scale poorly with problems that have large state spaces or non-Markov tasks, which are tasks where the state of the environment is only partially observable [18].

[can probably elaborate a bit more on the different types of tasks that can be accomplished here]

Neuroevolution (NE) is a method of artificially evolving Artificial Neural Networks (ANNs) by implementing Evolutionary Algorithms (EAs) to perform tasks such as training the connection weights and designing the network topology[13, 49].

By evolving a population of ANNs, NE searches through the space of possible controllers by evaluating each candidate solution using a fitness function and selecting the most successful individuals for reproduction, a process analogous to natural selection [18, 16].

This predetermined fitness function must be designed so that its implementation will guide the NE's search through the problem space, such that it indicates the desired behaviour from the ANN.

The different NE methods can be roughly split into two categories according to their encoding method, either direct or indirect. In direct encoding methods, each element in the genotype explicitly encodes an independent aspect of the phenotype, such that there is a one-to-one mapping between the genotype and the phenotype. In indirect (also called generative or developmental) encoding methods, each element in the genotype is implicitly described by a computable function allowing for a much more compact representation of the genotype. It also allows for genetic information to be reused [**Stanley2003**, 6].

The high level of complexity inherent in real life tasks tend to result in deceptive fitness landscapes [15]. These are fitness landscapes of multimodal problems that have numerous local optima [11]. In multimodal problems, it is possible for the NE search to get stuck around a local optima and lead to the NE method prematurely converging to a suboptimal solution [15].

For this reason, various NE search functions (also referred to as fitness functions) are investigated. The first search function being the objective search which simply evaluates the suitability of a solution by using an objective fitness function [27]. Novelty search is an approach that provides candidate solutions with a reward based how different their behaviour is from other candidate solutions [15]. The third approach is a hybrid search which attempts to combine EAs with a local search [**Castillo2007**].

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

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

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

The ability to automate construction tasks through evolving ANN controllers for multi-robot teams using EAs presents several considerable benefits. Automation such as this could be used to assist in projects such as producing low-cost housing and reduce high accident rates due to human error that accompany traditional construction methods [**Khoshnevis2003**]. It would be useful 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 difficult and often extremely dangerous [47].

This thesis is therefore focussed on investigating the effectiveness of various NE methods in solving collective behaviour tasks, with specific emphasis on the indirect encoding method called HyperNEAT [41], as well as the aforementioned search functions.

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

1.1.2 Areas of Research in Computer Science

Pattern Classification

'The task of pattern classification is to assign an input pattern (like speech waveform or handwritten symbol) represented by a feature vector to on of many prespecified classes. Well-known applications include character recognition, speech recognition, EEG waveform classification, blood cell classification, and printed circuit board inspection' [23].

Clustering/Categorization

'In clustering, also known as unsupervised pattern classification, there are no training data with known class labels. A clustering algorithm explores the similarity between the patterns and places similar patterns in a cluster. Well-known clustering applications include data mining, data compression, and exploratory data anyalysis ' [23].

Function Approximation

'Suppose a set of n labeled training patterns (input-output pairs), (x_1, y_1), (x_2, y_2),..., (x_n, y_n) have been generated from an unknown function $f(x)$ (subject to noise). The task of function approximation is to find an estimate, say f^1, of the unknown function f. Various engineering and scientific modeling problems require function approximation' [23].

(

Prediction/Forecasting) 'Given a set of n samples $y(t_1)$, $y(t_2)$,..., $y(t_n)$ in a time sequence, t_1, t_2,..., t_n' the task is to predict the sample $y(t_(n+1))$"

1.1.3 Possible Applications

Just a list of all the possible future applications for the research that is being performed in this investigation.

In this section, we go on to explain some possible applications of multi-robot teams that are able to solve the collective construction task, specificall:

Underwater Construction

-¿ Underwater construction: This is one of the most dangerous occupations in the entire world. Be sure to give some research related to studying the risks and the need for being able to perform underwater construction. Find some examples like the underwater roads and internet cables There could be a broader section title like "Construction in Dangerous Areas". This basically applies to

Space Exploration/Terraforming

-¿ Terraforming remote planets: Can mention something to do with all of the Mars missions being performed at the moment and how that could be the next frontier considering we are fucking up the Earth so much. This could allow us to jumpstart the terraforming process without needing to figure out how to get humans+resources+equipment on the planet. We can send a team of robots ahead while we figure things out here. We can send them up to a planet several years ahead

of us. These robots will arrive and begin to explore the planet, figuring out which resources are available and how they can be used to build any sort of structural habitat that would be safe for humans to arrive at in the future.

Production of Low Cost Housing

especially for remote rural areas where it is difficult for traditional construction equipment to gain access or to get construction materials transported there.

-¿ Production of low cost housing in remote and rural areas: This is an important aspect. Try find some examples of people needing to build some sort of housing in dangerous areas where you can't get normal construction equipment in to the area

-¿ Reduce human error: Prevent all damage that can be caused by human error. Loss of human life would also be minimised

-¿ Can maybe expand and take a look at using these types of ANN controllers for robots that can be used in mines for long periods of time. Reduce the loss of human life again

Example Search and Rescue Teams

This section could perhaps be moved to the Introduction part of the paper? It is an existing application but it is more an example reference for cooperative collective robotics like the robocup soccer.

These subsections are explaining other possible applications of the collective construction task specifically.

1.2 Problem Statement

Considering the dangers inherent in the environments in which these robots are meant to be implemented, these robot teams need to be able to handle/accommodate for any damage that it is likely to sustain.

The whole point of using these robots is because it is either too dangerous for humans or the location is too remote for conventional means of access. As such, these robots will have to be able to complete their assigned task and be able to either accommodate sustaining damage or be able to work around it since it will not be possible/feasible for humans to be able to perform any repairs.

Additionally, it would be most effective to have these robots continue working despite any damage they have sustained as it would not be too expensive to continue deploying new replacement robots.

–can maybe rephrase the bottom to make it more relevant to the context of the 'Research Aims' section In this study, the specific focus is on the evolution of morphologically robust behavior for robotic teams that must accomplish collective behavior tasks.

Given the added computational complexity of co-evolving body-brain couplings (maybe add a reference to the original mention of this back in the Introduction) for behaviorally and morphologically heterogenous robots, this study focuses on evolving collective behaviors for *homogenous* teams with a range of fixed morphologies.

–Spitballing ideas 'There are also other ways of compentaring for the loss of a sensor, particularly because the robot team is homogeneous. It could be possible that some of the other agents are able to accommodate for an agent losing some of its input sensors and its abilities becoming restricted. Lets say that one of the agents loses its construction sensor, it should still be able to assist other robots in carrying a resource

around (it can follow another robot that has a construction sensor). Although this may require that the individual robots are able to communicate with each other and indicate when they need assistance and in what capacity.'

Another alternative could be to have a separate robot team that is dedicated solely to performing repairs on the robots that are performing the collective construction task. But then you would also need to have a robot that is capable of performing repairs on the 'repair-robot' since it can also sustain damage, and this would go on recursively forever, needing a construction robot for a repair robot for a construction robot for a repair robot etc.

Since the controllers can't be trained from scratch to use the new sensor morphology and the structure of the ANN can not be altered on-the-fly, the existing controller must be robust in such a way as to be able to use its remaining sensors in order to complete the task at hand.

some possible future work: Can suggest looking into researching the possiblity of having a heterogeneous robot team in which some of the members are designated 'builders' and some are designated 'repairers'. The repairing robots can be very simple and basic and inexpensive and practically disposable. Their only purpose is to repair components on the 'builders' and would not need to be too complex at all (especially compared to the builder robots). Another approach would be to have a garage of sorts, which would be an extremely robust box with robot arms inside. The damaged builders can then enter the repair place which will then perform the necessary replacements etc. This garage will have to be very robust so as to be able to withstand the harsh conditions

1.3 Research Question

The research contribution is to demonstrate the efficacy of developmental neuroevolution (HyperNEAT) for addressing the evolution of morphologically robust controllers in the context of collective robotics. To address this objective, this study tested and evaluated five different robot team sensory configurations (morphologies) in company with three collective construction tasks of increasing complexity. The fittest *Artificial Neural Network* (ANN) controller evolved for a given morphology was then transferred and evaluated in each of the other team morphologies. Such transferred controllers were also evaluated in each of the three tasks. In all cases, evolved controller efficacy was evaluated in terms of collective construction task performance yielded by a robot team.

"Which implementation of the HyperNEAT algorithm is able to produce the most robust controllers in such a way that they are able to consistently solve a collective behaviour task of increasing complexity?"

1.3.1 Research Aims

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

- 'To investigate the behavioural robustness of multi-robot team controllers evolved using the HyperNEAT algorithm with respect sensory configurations when implemented in solvein 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: some complexity and no cooperation.
3. Level 3: some complexity and some cooperation.

We apply the HyperNEAT [40] neuro-evolution method to evolve behavior for a range of morphologically homogenous teams that must solve various collective construction tasks.

1.3.2 Research Contributions

The research contribution is to demonstrate the effectiveness of developmental neuro-evolution (HyperNEAT) for addressing the evolution morphologically robust controllers in the context of collective robotics.

In order to address this objective, this study tested and evaluated five different robot team sensory configurations (referred to as morphologies) in company (in conjunction?) with three collective construction tasks presenting increasing levels of task complexity. The fittest *Artificial Neural Network* (ANN) controller evolved for a given morphology was then transferred to each of the other morphologies and its task performance re-evaluated and compared repectively. Such transferred controllers were also evaluated in each of the three collective construction tasks (each one of increasing complexity). Each controller is only implemented in the task level for which it was evolved make sure that this is true. Was each controller evolved across all of the complexities or was each controller evolved on a per task basis? In all cases, evolved controller efficacy was evaluated in terms of the collective construction task performance yielded by a robot team

1.4 Existing Applications And Problem Spaces

In this section, we go on to outline and examine some of the existing applications that implement a similar solution approach along with the problem spaces that are being addressed by the different investigations.

[Be sure to provide a subsection for each approach that would demonstrate an important aspect of EC] RoboCup Soccer: cooperation and teamwork Collective Gathering: So similar to the collective construction task, indicating the efficacy

Make sure to try and find existing work where they use NE to co-evolve the different sensory configurations

In this section, we go on to outline the various implementations of Evolutionary Computing approaches in real-world examples. Some of these examples include the following

Be sure to outline the different algorithms and learning rules that were implemented in each of the approaches

Similar research includes investigating the ability of NE methods to find a minimal sensory configuration such that a team of homogenous robots are still able to complete a task requiring collective behaviour [43], as well as an investigation into evolving specialised sensor resolutions in a multi-agent task [34].

What makes this project different from existing research, is that it is specifically investigating the ability of HyperNEAT evolved controllers to deal with the loss of input sensors and still be able to solve cooperative tasks of increasing levels of complexity. The behavioural robustness of a controller will be tested by transferring an evolved controller from its original sensory configuration and implement it with each of the other sensory configurations in the simulation. Once this has been done, the controller is deployed into the simulated environment and its task performance is measured relative to its performance when using the morphology that it had been evolved with.

for each of the sections below outlining existing research, for each example: -¿ have a use case description, illustrating the scenarios that they are evaluating -¿ What contributions these researchers are making

1.4.1 Emergency Search-and-Rescue

[25] for the entire section below RoboCup Search and Rescue as a secondary domain for RoboCup activities 'some good examples of how these learned behaviours can be transferrable to other problem domains that require similar approaches' Disaster rescue is one of the most serious social issues which involves very large numbers of heterogenous agents in the hostile environment

Problem domains/use case scenario At 5:46AM, 17 January 1995, a large earthquake of a moment magnitude 6.9 hit Kobe city, Japan, killing over 6000 people and crushed almost 1/5th of all the housing structures [25]. Immediately after the earthquake, over 300 fires were reported and they started to quickly spread over wider areas. Fire fighting attempts were unsuccessful due to disruptions in the water supply. The earthquake had caused significant structural damage to local reservoirs, resulting in the water having leaked out within a few hours. To make situations even worse, debris from collapsed wooden houses and surrounding trees cluttered roads and other various, turning these would-be firebreaks into veritable combustion pathways. This debris caught fire while it was on the roads and the open space resulted in winds being channelled right over the fire, causing it to spread even faster along these open spaces and have easy access to the rest of the connected infrastructure. Additionally, this resulted in paramedics and emergency response teams not being able to reach disaster sites along conventional routes. Which also means that even if they were able to get human teams in there, it would be near impossible to transport the heavy equipment that would be needed (firetrucks, jacks and lifts, whatever tools they need to remove debris). Aerial surveillance was able to provide an overall view, but with very little details as to the situation on the ground level. The problem with using helicopters an planes is the noise that they create. This makes it very hard for emergency response teams to hear the faint sounds of people trapped underground and calling for help. These faint sounds that the victims make is the only source of information for these teams (at the time anyways, there may be some more advanced technology by now)

1.4.2 RoboCup Soccer

-cooperation between individuals

1.4.3 Collective Gathering

We chose to elaborate on this approach specifically because it is so similar to the construction task in question that we are investigating in this paper.

1.4.4 Balancing Pendulum

This is the classic problem space for controllers produced using Machine Learning approaches. (Not sure if this is necessarily Evolutionary Algorithms)

This is a good example to have because it is easily applied to increasing degrees of complexity The balancing pendulum problem has been implemented for both a single, double, and even a triple jointed pendulum [Can maybe find some pictures or GIFs of the controllers in action]

1.4.5 Space Antenna

This is some good research because it shows that what we think to be optimal design might not necessarily be that at all. It is a good example of how evolutionary computing approaches are able to find solutions that we as humans would never even have considered as being options.

Be sure to find the comparison examples: pictures of the antenna that the controller was able to create

Not too sure where to place this section/ Mention the previous research that has been conducted in this field -¿ RoboCup Soccer examples: find some research papers -¿ double and triple pendulum example: where the robot can balance the pole upright -¿ Collective gathering task -¿ Find that paper where they created the space antenna in the super weird shape

Chapter 2

Background Research

2.1 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 ' [8].

'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.' [8].

2.1.1 Architectures

An Aritificial Neural Network (ANN) is an abstracted and simplified model that represents the functioning of a biological brain [31]. A single ANN consists of numerous interconnected simple computational units, called neurons, that are ordered into layers so as to create a neural net [13].

An ANN consists of a set of processing elements, also known as neurons or nodes, which are interconnected [48].

An ANN can be described as a directed graph in which each node, i, performs a transfer function f_i of the form [48] (this is the sentence that originally introduced the function outlined here 2.1)

Each neuron consists of several inputs, some specific activation function and some output. An ANN receives input from the environment at its input layer [18].

Each neuron in a non-input layer then calculates the weight-ed sum of its inputs, referred to as the activation value [50, 13], 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 [50].

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 2.1, 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 2.1.

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 [51]. This makes ANNs a favourable choice for agent controllers in accomplishing various tasks [50].

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 [8, 49].

"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" [48]

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

2.2 Learning in ANN's

what is learning "An agent is learning if it improves its performance on future tasks after making observations about the world." [39]

"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." [39] "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" [39]. "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" [39].

'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: ' [39] 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.' [39].

'By seeing many camera images that it is told contain buses, it cn learn to recognize them (2)' [39].

'By trying actions and observing the results - for example, braking hard on a wet road - it can learn the effects of its actions (3)' [39].

'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)' [39]

there are no examples for the last 2 components...?

'There are 3 types of feedback that determine the three main types of learning: ' [39] -¿ '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 [48].

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 [48].

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 [48]:

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.

2.2.1 Supervised Learning

page 694-695 of the cited text

'The task of supervised learning is this: ' [39] (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 feed-forward 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"

Classification

'Predict a discrete target variable'

Regression

'Predict a continuous target variable'

2.2.2 Unsupervised Learning

Clustering

Dimensionality Reduction

2.2.3 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.'[24] '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' [24] -¿ 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' [24]. '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' [24]. 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 [24].

'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' [24]

'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 [24]:

- 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' [24]

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 ' [24]

'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' [24]

'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.' [24]

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' [24].

The Finite Horizon Model

[24] '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:'

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

2.3 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)" [14]

'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"' [14].

should probably just make sure of the content that I am using from [14] 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"

2.4 Evolutionary Computing / Evolutionary Algorithms

for the basic outline below, this was taken directly from the outline in [48], 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

the background sound in the Eminem's song W.T.P is the same soundtrack clip that they used in the first Hulk movie with Eric Bana

2.5 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

2.5.1 Exploitation vs Exploration

REINFORCEMENT LEARNING PAPER [24] 'One major difference between reinforcement learning and supervised learning is that a reinforcement learner must explicitly explore its environment.' [24] 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' [24]

'The simplest possible Reinforcement Learning problem is known as the $k-armed$ bandit problem' [24] '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?'[24]. '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'

2.6 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.' [32]

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 [32].

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.

2.6.1 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' [32]

'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' [32].

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.' [32].

'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' [32].

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' [32].

'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.' [32].

'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.' [32].

'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 ' [32].

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

2.6.2 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'

find some more references and information on the balancing pendulum problem as this is referred to as a benchmark task

'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 ' [32].

'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] ' [32].

'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] ' [32].

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' [32].

FROM THE LITERATURE REVIEW VERBATIM

Neuroevolution (NE) provides a way of combining EAs and ANNs [13]. It uses EAs to evolve the connection weights, topologies and activation functions of ANNs in order to learn a specific task or behaviour [17]. NE searches for an ANN with optimal performance based on a fitness function that determines an ANN's suitability to performing a task [13].

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 [18].

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 [16, 49].

In a standard RL scenario an ANN interacts with its environment in discrete time steps [22]. 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 [33].

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 [13].

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

2.7 NEAT

2.8 HyperNEAT

2.9 Fitness Functions

2.9.1 Objective Fitness

2.9.2 Novelty Fitness

2.9.3 Hybrid Fitness

2.10 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 [35].

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?) [35].

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'

2.10.1 Collective Gathering Task

2.10.2 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

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] Y. Asai and T. Arita. "Coevolution of Morphology and Behavior of Robots in a Multiagent Environment". In: *Proceedings of the SICE 30th Intelligent System Symposium*. Tokyo, Japan: The Society of Instrument and Control Engineers, 2003, pp. 61–66.

[2] J. Auerbach and J. Bongard. "Environmental Influence on the Evolution of Morphological Complexity in Machines". In: *PLoS Computational Biology* 10(1) (2014), e1003399. doi:10.1371/journal.pcbi.1003399.

[3] 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.

[4] G. Buason, N. Bergfeldt, and T. Ziemke. "Brains, bodies, and beyond: Competitive co-evolution of robot controllers, morphologies and environments". In: *Genetic Programming and Evolvable Machines* 6(1) (2005), pp. 25–51.

[5] N. Cheney et al. "Unshackling Evolution: Evolving Soft Robots with Multiple Materials and a Powerful Generative Encoding". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Amsterdam, Netherlands: ACM Press, 2013, pp. 167–174.

[6] Jeff Clune et al. "On the performance of indirect encoding across the continuum of regularity". In: *Evolutionary Computation, IEEE Transactions on* 15.3 (2011), pp. 346–367.

[7] D. D'Ambrosio and K. Stanley. "Scalable Multiagent Learning through Indirect Encoding of Policy Geometry". In: *Evolutionary Intelligence Journal* 6.1 (2013), pp. 1–26.

[8] Judith E Dayhoff and James M DeLeo. "Artificial neural networks". In: *Cancer* 91.S8 (2001), pp. 1615–1635. ISSN: 1097-0142. DOI: 10.1002/1097-0142(20010415)91:8+<1615::AID-CNCR1175>3.0.CO;2-L. URL: http://dx.doi.org/10.1002/1097-0142(20010415)91:8+{\textless}1615::AID-CNCR1175{\textgreater}3.0.CO;2-L.

[9] S. Doncieux et al. "Evolutionary robotics: what, why, and where to". In: *Frontiers in Robotics and AI — Evolutionary Robotics* 2(4) (2015), pp. 1–18.

[10] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Berlin, Germany: Springer, 2003.

[11] Agoston E Eiben and James E Smith. "Introduction to evolutionary computing". In: 53 (2003).

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

[13] Joydeep Ghosh, Benjamin J Kuipers, and Raymond J Mooney. "Efficient evolution of neural networks through complexification". In: (2004).

[14] David E Goldberg and John H Holland. "Genetic algorithms and machine learning". In: *Machine learning* 3.2 (1988), pp. 95–99.

[15] 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.

[16] Faustino J Gomez, Doug Burger, and Risto Miikkulainen. "A neuro-evolution method for dynamic resource allocation on a chip multiprocessor". In: *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on.* Vol. 4. IEEE. 2001, pp. 2355–2360.

[17] Faustino J Gomez and Risto Miikkulainen. "Solving non-Markovian control tasks with neuroevolution". In: *IJCAI.* Vol. 99. 1999, pp. 1356–1361.

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

[19] Matthew Hausknecht et al. "HyperNEAT-GGP: A HyperNEAT-based Atari general game player". In: *Proceedings of the 14th annual conference on Genetic and evolutionary computation.* ACM. 2012, pp. 217–224.

[20] J. Hewland and G. Nitschke. "Evolving Robust Robot Team Morphologies for Collective Construction". In: *The Benefits of Adaptive Behavior and Morphology for Cooperation in Robot Teams.* Cape Town, South Africa: IEEE, 2015, pp. 1047–1054.

[21] G. Hornby and J. Pollack. "Creating High-Level Components with a Generative Representation for Body-Brain Evolution". In: *Artificial Life* 8(3) (2002), pp. 1–10.

[22] Christian Igel. "Neuroevolution for reinforcement learning using evolution strategies". In: *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on.* Vol. 4. IEEE. 2003, pp. 2588–2595.

[23] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. "Artificial neural networks: A tutorial". In: *Computer* 29.3 (1996), pp. 31–44.

[24] L. Kaelbling, M. Littman, and A. Moore. "Reinforcement learning: A survey." In: *Journal of Artificial Intelligence* 4.1 (1996), 237–285.

[25] H. Kitano et al. "RoboCup Rescue: Search and Rescue in Large-Scale Disasters as a Domain for Autonomous Agents Research". In: *Proceedsings of the Conference on Systems, Man and Cybernetics.* ACM Press.

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

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

[28] H. Lipson and J. Pollack. "Automatic design and manufacture of robotic life forms". In: *Nature* 406.1 (2000), pp. 974–978.

[29] H. Lund. "Co-evolving Control and Morphology with LEGO Robots". In: *Morpho-functional Machines: The New Species (Designing Embodied Intelligence).* Ed. by H. Fumio and R. Pfeifer. Berlin, Germany: Springer, 2003, pp. 59–79.

[30] C. Mautner and R. Belew. "Evolving robot morphology and control". In: *Artificial Life and Robotics* 4(3) (2000), pp. 130–136.

[31] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[32] R. Miikkulainen. "Neuroevolution". In: *Encyclopedia of Machine Learning*. Ed. by C. Sammut and G. Webb. New York, USA: Springer, 2010, pp. 716–720.

[33] Risto Miikkulainen. "Evolving Neural Networks". In: *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*. GECCO '10. New York, NY, USA: ACM, 2010, pp. 2441–2460. ISBN: 978-1-4503-0073-5. DOI: 10.1145/1830761.1830902. URL: http://doi.acm.org/10.1145/1830761.1830902.

[34] G. Nitschke, M. Schut, and A. Eiben. "Collective Neuro-Evolution for Evolving Specialized Sensor Resolutions in a Multi-Rover Task". In: *Evolutionary Intelligence* 3(1) (2010), pp. 13–29.

[35] 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.

[36] Geoff S Nitschke, Martijn C Schut, and A E Eiben. "Evolving behavioral specialization in robot teams to solve a collective construction task". In: *Swarm and Evolutionary Computation* 2 (2012), pp. 25–38.

[37] R. O'Grady, A. Christensen, and M. Dorigo. "SWARMORPH: Morphogenesis with Self-Assembling Robots". In: *Morphogenetic Engineering, Understanding Complex Systems*. Ed. by R. Doursat. Berlin, Germany: Springer-Verlag, 2012, pp. 27–60.

[38] M. Rubenstein, A. Cornejo, and R. Nagpal. "Programmable Self-Assembly in a Thousand-Robot Swarm". In: *Science* 345(6198) (2014), pp. 795–799.

[39] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[40] 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.

[41] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. "A hypercube-based encoding for evolving large-scale neural networks". In: *Artificial life* 15.2 (2009), pp. 185–212.

[42] P. Verbancsics and K. Stanley. "Constraining connectivity to encourage modularity in HyperNEAT". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2011, pp. 1483–1490.

[43] J. Watson and G. Nitschke. "Deriving Minimal Sensory Configurations for Evolved Cooperative Robot Teams". In: *Proceedings of the IEEE Congress on Evolutionary Computation*. Sendai, Japan: IEEE, 2015, pp. 3065–3071.

[44] J. Watson and G. Nitschke. "Evolving Robust Robot Team Morphologies for Collective Construction". In: *Proceedings of the IEEE Symposium Series on Computational Intelligence*. Cape Town, South Africa: IEEE, 2015, pp. 1039–1046.

[45] J. Werfel and R. Nagpal. "Three-Dimensional Construction with Mobile Robots and Modular Blocks". In: *The International Journal of Robotics Research* 27(3-4) (2008), pp. 463–479.

[46] J. Werfel, K. Petersen, and R. Nagpal. "Designing Collective Behavior in a Termite-Inspired Robot Construction Team". In: *Science* 343(6172) (2014), pp. 754–758.

[47] 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.

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

[49] Xin Yao. “Evolving artificial neural networks”. In: *Proceedings of the IEEE* 87.9 (1999), pp. 1423–1447.

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

[51] Guoqiang Zhang, B Eddy Patuwo, and Michael Y Hu. “Forecasting with artificial neural networks:: The state of the art”. In: *International journal of forecasting* 14.1 (1998), pp. 35–62.