

https://dspace.org/

Evolving soft robots with CPPN-NEAT in a randomised domain with realistic fluidic elastomer actuators

Pienaar, Michael

2024
https://hdl.handle.net/10539/37395

Downloaded from a DSpace repository. https://dspace.org/

Evolving Soft Robots with CPPN-NEAT in a Randomised Domain with Realistic Fluidic Elastomer Actuators

Michael Pienaar

A dissertation submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, March 2023

DECLARATION

I declare that this dissertation is my own, unaided work, except where otherwise acknowledged. It is being submitted for the degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination at any other university.

Signed this 1st of March 2023



Michael Pienaar

To my parents, Jean and Nick, my brother, Andrew, and my partner, Alessandra for their unconditional love and support.

ACKNOWLEDGEMENTS

I would like to the thank the following people:

My parents for their unbelievable love and support through my journey of achieving my goals. You constantly set an example to me with your work ethic and attitude towards life.

My brother for his love and belief in me, giving me the courage to tackle any challenge I face throughout life.

My beloved girlfriend, Ally, for your endless love, support, and encouragement. Your energy and drive blows me away and fuels me to be the best I can be.

The employees of Inria and SOFA for their incredible advice and work they do to help others use their software to build the fields of soft robotics and medical simulation.

Lastly, I would like to thank my supervisor, Dean, for excellent advice throughout this research, and for consistently taking time out to discuss my work and progress on a weekly basis.

ABSTRACT

Robotics is becoming more and more integrated into our lives; however, there are limitations to what can be achieved using traditional robots. Traditional robots perform well in closed environments for repetitive tasks but underperform in unknown, open environments. Additionally, they can potentially damage animals, people, and environments around them and are very expensive to design and manufacture. In contrast, soft robots are inherently safe, are cheap to make and are excellent at adapting to variations in their environment. This makes soft robots more suitable than rigid robots for medical applications, hospitality, research and exploration in natural environments, and extra-terrestrial exploration. Unfortunately, soft robots are difficult to design due to the nonlinearity of their behaviour.

Previous research has shown an evolution strategy, CPPN-NEAT, could be used to design both the morphologies and controllers of virtual soft robots. However, these studies do not accurately represent real soft robots and real environments, such that their results have no real-world applicability. In this research, the gap between evolving virtual soft robots and soft robots with real applicability is reduced by using a more realistic simulation environment, SOFA, realistic fluidic elastomer actuators in the evolution, implementing domain randomisation during the evolution, and lastly, by growing soft robots from central mesh like what occurs in developmental biology.

It was successively shown that the entire structure and composition of soft robots that use fluidic elastomer actuators can be evolved in SOFA. Interestingly, with these improvements, designs that resembled real soft robotic designs were evolved showing the realism of the environment and set-up. Furthermore, it was shown how domain randomisation can improve the evolutions' ability to find soft robots that can handle unknown environments better. Lastly, soft robots were successively evolved by growing them from central elements, which in turn expanded the possible sizes and shapes of the soft robots.

CONTENTS

Declaration i

Acknowledgements i

Abstract ii

Contents iii

List of Figures vi

List of Tables x

1 Introduction 1

1.1 Background and Motivation 1

1.2 Problem Statement 2

2 Literature Review 3

2.1 Evolutionary Computation 3

2.1.1 Neuroevolution of Augmenting Topologies (NEAT) 4

2.1.2 Evolutionary Robotics 6

2.2 Genetic Representation of Evolved Robots 7

2.2.1 Direct Encoding 7

2.2.2 Directed Graphs 7

2.2.3 Gene Regulatory Networks 8

2.2.4 Compositional Pattern Producing Networks 10

2.3 Search Algorithms 11

2.3.1 Fitness Search 11

2.3.2 Novelty Search 12

2.3.3 Quality-Diversity Algorithms 12

2.4 Soft Robots 13

2.4.1 Introduction 13

2.4.2 Types of Soft Robotic Actuators 14

2.5 Simulation Environments for Evolving Soft Robots 20
2.5.1 VoxCad 20
2.5.2 Simulation Open Framework Architecture (SOFA) 21
2.6 Domain Randomisation 23
2.7 Related Work 24
2.7.1 Evolving Rigid Robots in Silico 24
2.7.2 Evolving Soft Robots in Silico 29
2.7.3 Evolving Real Robots 33
2.8 Identified Gaps in Literature 34
3 Research Scope and Overview 35
3.1 Research Objectives 35
3.2 Research Scope and Limitations 35
3.3 Research Contributions 36
3.4 Dissertation Layout 36
4 REsearch Methodology 37
4.1 Introduction 37
4.2 Validation of Modifications 37
4.2.1 Modification to NEAT Initialisation Settings 37
4.2.2 Modified Fitness Function 40
4.3 Effect of Domain Randomisation on Performance and Diversity 43
4.4 Increasing the Solution Space through Mesh Growth 45
4.5 Comparison between VoxCad and SOFA 46
5 Experimental Set-up 47
5.1 Overview of the Set-up and Procedure 47
5.2 Input Methodology 48
5.3 CPPN-NEAT Implementation 50
5.4 Mesh File Generation 51
5.5 Set-up of Simulation Environments 53

5.5.1 Set-up of VoxCad 53
5.5.2 Set-up of SOFA 53
5.5.3 Material Properties 55
5.5.4 Domain Randomisation 55
6 Results and Discussion 56
6.1 Validation of Modifications 56
6.1.1 NEAT Settings and Fitness Function used in this Work 56
6.1.2 NEAT Settings used in Previous Literature 58
6.1.3 Fitness Function used in Previous Literature 62
6.2 Effect of Domain Randomisation on Performance and Diversity 66
6.3 Increasing the Solution Space Through Mesh Growth 78
6.4 Comparison between VoxCad and SOFA 82
Conclusions 84
7 Recommendations for Future research 85
Refernces 86

LIST OF FIGURES

Figure 1: Example of the NEAT algorithm in action. 5

Figure 2: Example of a direct encoding for a soft robot 7

Figure 3: Directed Graph used to encode rigid structures. [18] 8

Figure 4: The genome structure used for in GRNs. [2] 9

Figure 5: Cable driven actuators used to control a gripper and an octopus-inspired soft robotic arm. 15

Figure 6: McKibben Actuator [40] 16

Figure 7: STIFF-FLOP Actuator consisting of multiple McKibben Actuators connected radially [40] 16

Figure 8: Pneumatic Network (PneuNet) actuator [42] 17

Figure 9: Vaccuum-powered soft actuators used for (a) linear [45] and (b) twisting actuation [46]. .. 18

Figure 10: Examples of DEAs 19

Figure 11: An example of a scene graph in SOFA 22

Figure 12: Method of reproduction used to combine two directed graphs in Sims' work. [18] 25

Figure 13: Top-performing Robots Evolved in Sims' Work. [18] 26

Figure 14: The top-performing robots evolved in Auerback and Bongard's work. [28] 27

Figure 15: The top-performing robots in Auerbach and Bongard's work evolved in (a) a simple environment and (b) a more complex environment. [49] 28

Figure 16: The modules used by the evolution to create robots – (a) a rigid module, (b) the central housing, and (c) an actuated hinge joint. [50] 29

Figure 17: Examples of the evolved modular robots using RoboGen framework [51] and Lamarckian evolution. [50] 29

Figure 18: An example of an evolved robot in Reiffel et al.'s work. [52] 30

Figure 19: Showing that equilateral tetrahedra cannot fill 3-D space. [53] 30

Figure 20: Some of the soft robots evolved in Cheney et al.'s work [1] 31

Figure 21: Evolved soft robots using GRN-NEAT. [2] 32

Figure 22: Examples of the achieved robotic behaviours and morphologies evolved in this work. [56] 34

Figure 23: Example of a heatmap showing the exploration of the solution space in terms of CPPN structure. 40

Figure 24: Visualisation of how the modified fitness function is calculated and how it would penalise inconsistent motions. 42

Figure 25: Examples of the unseen terrain for testing soft robots' ability to adjust to new environments. 44

Figure 26: An example of how a heatmap can be used to observe the exploration of the solution space. 45

Figure 27: Methods Diagram 47

Figure 28: The scene graph of simulation environment used to evaluate soft robots during the evolutionary process. 54

Figure 29: The motion of the top performing soft robots evolved from 5 independent runs, where each frame is captured at 0.3 second intervals. Videos for each can be found in the Digital Appendix. 57

Figure 30: The motion of the top performing soft robots evolved from 5 independent runs using NEAT settings from previous literature, where each frame is captured at 0.3 second intervals. Videos for each can be found in the Digital Appendix. 59

Figure 31: The CPPNs with the minimum and maximum complexities evolved using the modified NEAT settings and the NEAT settings used in previous work 60

Figure 32: Comparison of the amount of explored CPPN structures as a result of NEAT settings used in previous research versus that used in this research. 61

Figure 33: Heatmaps showing the exploration of the solution space in terms of CPPN structures as well as the performance of different structures. (a) The exploration of the evolution using settings from previous literature. (b) The exploration of the evolution using the settings in this research. 61

Figure 34: The total distance achieved by the highest fitness soft robots for 5 independent runs for both NEAT Settings used. The solid lines represent the median distances, and the dashed lines represent the ±95% bootstrapped confidence intervals. 62

Figure 35: The motion of the top performing soft robots evolved from 5 independent runs using the fitness function used in previous literature, where each frame is captured at 0.3 second intervals. Videos for each can be found in the Digital Appendix. 63

Figure 36: The speed of the soft robots during the evolution for 5 independent runs. (a) Using total distance as the fitness and (b) using the average of the lowest two distances travelled during 5 actuation cycles as the fitness. 64

Figure 37: Top-view showing the paths taken by the soft robots during 15 actuation cycles 65

Figure 38: The median distance travelled by the top-performing soft robots using the two fitness functions. The solid line represents the median value, and the dotted lines represent the 95% bootstrapped confidence intervals. 66

Figure 39: The motion of the top performing soft robots evolved from 5 independent runs using fitness search. The first frame shows the initial shape, thereafter, each frame is captured at 0.5 second intervals towards the end of the simulation. Videos for each can be found in the Digital Appendix. 67

Figure 40: The motion of the top performing soft robots evolved from 5 independent runs using novelty search. The first frame shows the initial shape, thereafter, each frame is captured at 0.5 second intervals towards the end of the simulation. Videos for each can be found in the Digital Appendix. 69

Figure 41: The motion of the top performing soft robots evolved from 5 independent runs using fitness search with domain randomisation. The first frame shows the initial shape, thereafter, each frame is captured at 0.5 second intervals towards the end of the simulation. Videos for each can be found in the Digital Appendix. 70

Figure 42: The median distance travelled by the top-performing soft robots. The solid line represents the median value, and the dotted lines represent the 95% bootstrapped confidence intervals. 71

Figure 43: The performance of the top performing soft robots from each search method on a sloping surface. The surface is constructed using the formula: $y = mr2$. 72

Figure 44: The performance of the top performing soft robots from each search method on a bumpy surface. The surface is constructed using the formula: $y = 0.1\sin\omega r$. 73

Figure 45: Material Composition: (a) Fitness Search (b) Novelty Search (c) Domain Randomisation 74

Figure 46: Heatmaps showing the median exploration of material compositions for each method: (a) fitness search, (b) novelty search, and (c) domain randomisation. 75

Figure 47: Heatmaps showing the median exploration of actuator 1 positions for each method: (a) fitness search, (b) novelty search, and (c) domain randomisation. 77

Figure 48: Heatmaps showing the median exploration of actuator 2 positions for each method: (a) fitness search, (b) novelty search, and (c) domain randomisation 77

Figure 49: The motion of the top performing soft robots evolved from 5 independent runs using the growing mesh method. The first frame shows the initial shape, thereafter, each frame is captured at 0.5

second intervals towards the end of the simulation. Videos for each can be found in the Digital Appendix..... 79

Figure 50: The frequency distribution of the length of soft robots from 5 evolutionary runs. 80

Figure 51: The frequency distribution of the width of soft robots from 5 evolutionary runs. 80

Figure 52: The frequency distribution of the height of soft robots from 5 evolutionary runs. 81

Figure 53: The median distance travelled by the top-performing soft robots. The solid line represents the median value, and the dotted lines represent the 95% bootstrapped confidence intervals..... 82

Figure 54: Comparison in the behaviour of soft robots evolved in VoxCad inside VoxCad versus SOFA. Videos for each can be found in the Digital Appendix..... 83

LIST OF TABLES

Table 1: Example of NEAT genome. 4

Table 2: An example of a directly encoded genome used in this work. [56] 33

Table 3: The evolutionary settings used in this research. 38

1 INTRODUCTION

1.1 Background and Motivation

Huge advancements in the design and control of traditional robots, have allowed robots to perform challenging tasks with extremely high accuracy, but only when the task is repetitive and is performed in a closed environment. Unfortunately, traditional robots underperform and even fail when the task varies slightly and the environment changes or is unknown. This inability of traditional robots to adapt to changes in their working environment is largely due to the materials that they are made from – rigid materials. Rigid materials inherently lack compliance to adapt to variations around them. These materials also makes it difficult to integrate sensing and actuation seamlessly into the materials themselves.

To overcome these challenges, it becomes useful to take inspiration from nature. Throughout the natural world, creatures are made from a combination of both soft and rigid material with sensors, actuators and control systems distributed throughout the body resulting in a complex system capable of a wide variety of tasks and adapting to these tasks and environment. One of the most dexterous and versatile creatures is the octopus. The octopus can move and sense with each arm independently, can perform hundreds of different tasks, and adapts extremely well to its environment, yet the octopus is completely made from soft and flexible materials apart from its beak. Utilising these same ideas in robotics could lead to similar performances in robots and is the primary motivation for the growing interest and research in the field of soft robotics. Robots are becoming more and more integrated into the world we live in, in the production line at work and the natural world for research. It is therefore becoming more important that these robots are safe to be around and they that they do not damage the environment in which they operate. Traditional rigid robots can be dangerous to work around due to the materials they are made of, whereas soft robots are inherently safe. Lastly, most soft robots are significantly cheaper to make than traditional robots, making them a great alternative for most applications.

One disadvantage of soft robots is that the properties that make them advantageous over traditional rigid robots, also make their behaviour substantially more difficult to model, predict, and control. Over the past three decades, significant progress has been made in evolutionary computation. Since evolutionary strategies excel in solving challenging problems where the solution space is complex, they are capable of evolving virtual robots and more importantly virtual soft robots. Unfortunately, evolution requires many generations to find solutions, which can only be done efficiently in virtual environments. A common problem in transferring solutions from simulation to the real world, known as the sim-to-real transfer, is the reality gap, where solutions designed in a virtual environment cannot handle the

variability and unpredictability of the real world. This work aims to improve the methods and tools used to overcome this reality gap and make soft robots that can be manufactured and used in the real world.

1.2 Problem Statement

Traditional rigid robots are more expensive and lack the adaptability and safety of soft robots, making soft robots a better alternative for many applications. Due to the complexity and unpredictability of soft, flexible materials, there are limited methods of designing and controlling soft robots as there are with traditional rigid robots. Cheney et al.[1] and Joachimczak et al.[2] have successively shown how evolutionary computation can be used to create virtual soft robots, but these soft robots cannot be made and used in the real world. This is caused by three limitations of previous work. Firstly, the focus was made on the speed of simulations as opposed to realism. Secondly, the actuation techniques used in the evolved soft robots do not correlate to existing actuators used in soft robots. Lastly, no method has been used to close the reality gap and improve sim-to-real transfer. Furthermorein Cheney et al.[1], the size of the evolved soft robots are constrained to a cube of voxels, which limits the possible designs that can be found, which in turn could limit the performance.

2 LITERATURE REVIEW

2.1 Evolutionary Computation

The countless inventions which have been inspired by nature. Examples include Velcro [3], the skin of marine vessels [4], the design of bullet trains [5], and even neural networks [6]. Throughout the natural world, you can find brilliant designs to aid in a creature's survival, and this is all thanks to the process of evolution which has occurred over billions of years. However, what if instead of taking the designs from nature, the process of evolution is mimicked itself, and this is exactly what evolutionary computation attempts to do [7]. To understand how the process of evolution can be used in computational problems, it is vital to first understand the mechanisms of natural evolution.

There are three core mechanisms underlying biological evolution that make it so successful in finding optimal designs and behaviours – natural selection, crossover, and mutation. Natural selection is most suitably simplified as the survival of the fittest - organisms with better designs and behaviours are more likely to survive and reproduce, passing their beneficial genes onto the next generation. The process of passing genes from one generation to another occurs through reproduction, but more specifically, crossover. Crossover is the process of exchanging homogenous genes between the parent genomes to combine potentially beneficial characteristics of both parents. In addition to crossover, mutations can also occur during reproduction. Mutation is the random process by which genes could be changed, added, or removed. These two processes result in offspring with a unique genotype and phenotype with some of the successful features and traits of its parents but potentially with new characteristics which could potentially improve its performance and increase its chance of survival even further [8].

Similarly, if a solution to a problem can be represented using a string of bits or any other data structure (the equivalent to genes) and the solution could be evaluated in some manner, then the methods of selection, crossover and mutation could be computed on these genes, to optimise the solutions over time. These methods of computing the processes of evolution, as well as how to represent solutions genetically differ throughout literature. However, the simplest method entails representing a possible solution as a string of bits with a fixed length. An initial population of solutions are created by randomly generating strings of bits. These solutions can then be evaluated and given a fitness based on their performance. The generated solutions with the highest fitness values are selected to create the next generation of solutions through crossover and mutation. Crossover is achieved by swapping segments of bits between two different solutions and mutation is achieved by simply changing single bits from a 1 to a 0 or vice versa. The new generation of solutions are evaluated, and the process is repeated until an optimal solution is found, or convergence is reached [7].

However, most solutions to problems cannot simply be represented as a string of bits. One such case of interest is the structure and weights of artificial neural networks (ANNs). ANNs have been at the

forefront of success in the artificial intelligence field with breakthroughs in computer vision [9, 10], natural language processing [11, 12, 13], and robotic controllers [14] every year. Although optimisation methods have been developed to optimise the ANNs, their performance has been shown to be dependent on their designed structure. Neuroevolution, the method of evolving ANNs, provides an alternative method to optimise not only connection weights within ANN but the structure as well [15].

2.1.1 Neuroevolution of Augmenting Topologies (NEAT)

Initially, the field of neuroevolution focused on optimising the connection weights of ANNs whilst keeping the neural network structure fixed, since a fully connected neural network can, in theory, approximate any continuous function [16, 17]. However, it has since been shown that optimising both the structure and connection weights can reduce the evolution time required to find an optimal solution by minimising the dimensionality of the search space. To evolve ANN's structure introduces more challenges, namely, how can the structure of the neural network be represented genetically, such that crossover and mutation can be performed constructively. Currently, one of the best neuroevolution methods, Neuroevolution of Augmenting Topologies (NEAT), provides solutions to this challenge in interesting ways [15].

In the NEAT method, the genome of each neural network contains a list of node genes and a list of connection genes (Table 1). Each node gene contains its node number, what type of node it is (hidden or output) and other properties of the node that could be evolved, such as the activation function. Each connection gene contains the input node number, the output node number, the weight of the connection, whether the connection is enabled and lastly, its innovation number. The first challenge to address is how to implement crossover between genomes, which may have different lengths and different structures, constructively. The solution lies in the use of innovation numbers and node numbers. When a new node or connection is added to the network, a new gene is placed at the end of the genome with the next available node or innovation number. This provides a way of tracking the origin of genes throughout the evolutionary process, thus genes with the same innovation or node number can be considered homologues. This ensures that crossover only occurs between genes of the same type, as seen in nature [15].

Node	Label	1	2	3	4	5		
	Type	Input	Input	Input	Output	Hidden		
Connection	Input	1	2	3	2	5	1	4
	Output	4	4	4	5	4	5	5
	Weight	0.7	-0.5	0.5	0.2	0.4	0.6	0.6
	Enabled	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE
	Innovation Number	1	2	3	4	5	6	11

Table 1: Example of NEAT genome.

NEAT begins with simple networks with no hidden nodes and grows and optimises during the evolutionary process through structural mutations. There are three different types of mutations: mutation of connection weights, adding or removing nodes, and adding or removing connections. Connection weight mutations are implemented by simply altering one or more of the connection weights randomly. New connections are added between two randomly selected nodes and a corresponding connection gene with a new innovation number is appended to the end of the genome. A new node is added by first disabling a randomly selected connection and adding two new connections between the added node and the other two nodes. Importantly, the connection providing the input to the new node is given a weight of 1 and the other connection is given a weight equal to that of the old connection. This minimises the initial impact of the new node [15]. Figure 1 shows this process.

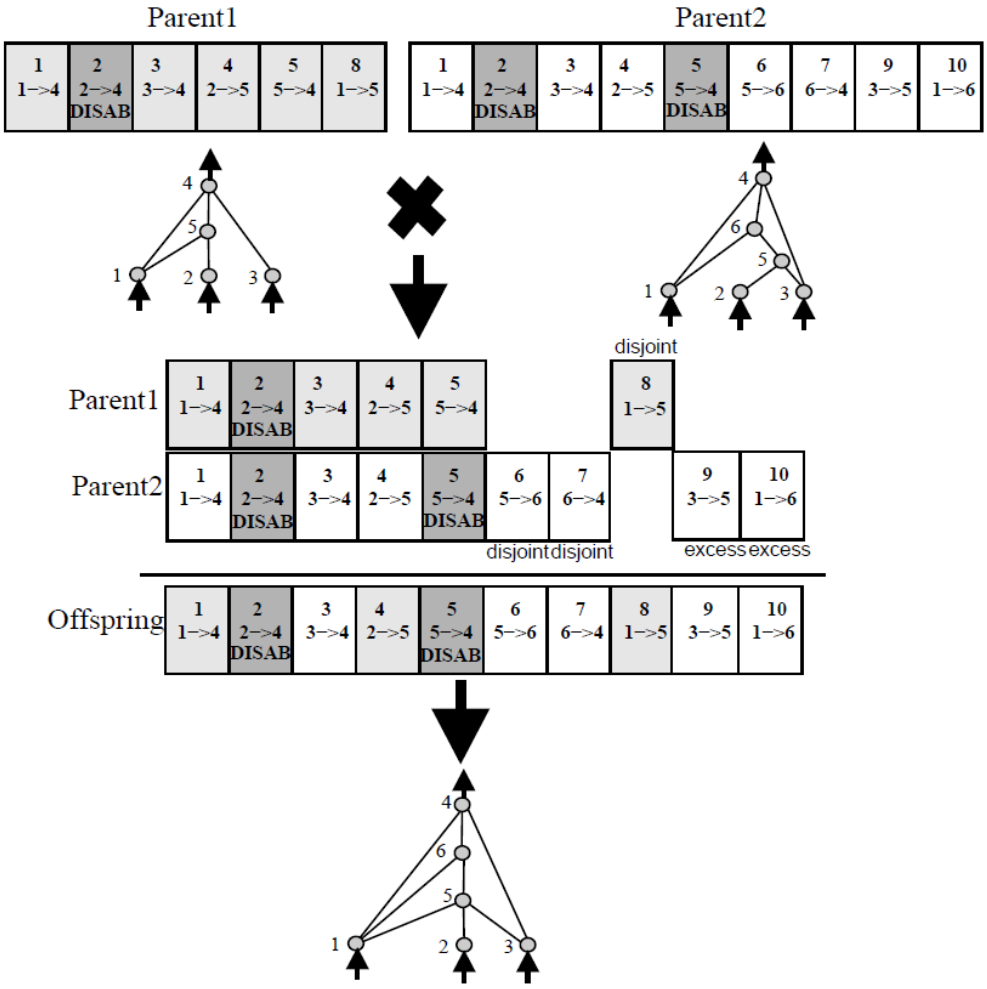


Figure 1: Example of the NEAT algorithm in action.

One of the major drawbacks of introducing structural changes to neural networks during the evolution, is that these structural mutations will initially be harmful to the performance of the ANN, thus unlikely to be selected to produce the next generation. Since most structural changes will be initially detrimental to the ANNs performance, the networks will be unlikely to grow and increase in complexity even though

this may be necessary to achieve higher fitness values. By once again taking inspiration from nature, NEAT uses speciation to protect innovative structural changes. Utilising innovation numbers one can compare the structure between two neural networks and determine how many unique nodes and connections there are in each genome, which indicates how different the two genomes are. If the genomes are sufficiently different, they can be separated into different species, referred to as speciation. This is done by calculating the compatibility distance between one genome and another, if the compatibility distance is above a predetermined value, then the two genomes are placed in two different species. If the compatibility distance is below the threshold, then the two genomes are placed in the same species. Speciation ensures that sufficiently different genomes are given time to optimise, as the same proportion of genomes is selected from each species during reproduction, irrespective of fitness [15].

One potential issue with speciation is that it not only protects innovative mutations but also mutations that do not provide any advantage to the network. To overcome this, if a species has not improved in performance after a certain number of generations, referred to as stagnation, the species is removed from the population. Additionally, to allow all species to evolve effectively, it must be ensured that one or two species do not dominate and become overpopulated. This can be done by implementing explicit fitness sharing, where genomes within a species share the fitness within their species. This ensures that even if the genome performs well if there are too many in its species it will get a lower fitness value[15].

2.1.2 Evolutionary Robotics

In the field of robotics, evolutionary computation has mostly been used to optimise the control of robots. However, in 1994 Karl Sims showed that it was possible to evolve not only the controller of a virtual robot but also the morphology of the robot. Although optimising the controller of a robot can greatly improve the performance of a robot, it will always be limited by its design and structure. The natural world is a perfect example of what can be achieved when evolving both the brain and morphology of a creature [18].

Despite the success achieved by Karl Sims and successive studies thereafter, evolutionary robotics became redundant due to advancements in robotic design and modelling techniques. These advancements have allowed engineers to design incredibly advanced robots with excellent performance. However, with the growing interest in soft robotics, and the advancement being made in evolving virtual soft robots, the evolutionary computation could become a vital component in the process of designing future robots. The primary reason evolutionary computation may be essential is that can be difficult to model mathematically, making them difficult to design using traditional techniques. Evolutionary computation on the other hand excels at finding optimal solutions when the search space is large, and the problem is complex [1].

To evolve soft robots, there are two core components that affect the success of evolution at finding feasible soft robots – how the soft robots are encoded genetically and how the solution space is explored.

2.2 Genetic Representation of Evolved Robots

Genetic representation is important in two ways. First, solutions need to be represented accurately such that changes to the genome represent changes to the performance of the solution. Secondly, an efficient genetic encoding can simplify the solution space using abstract representations of the solution, allowing the evolutionary algorithm to find optimal solutions in less time. In the human body, the complexity and diversity of trillions of structures are all encoded using only 30 thousand genes, which highlights the importance of an efficient genetic encoding [19].

2.2.1 Direct Encoding

This is the simplest form of genetic encoding where the solution is directly encoded into the genome. A direct encoding for a soft robot can be implemented by encoding each voxel into the genome. Each voxel gene can contain whether the voxel is activated and what material properties it has as shown in Figure 2. An important characteristic of direct encodings to note is that the dimensions of the search space of the genome are equal to the number of all possible compositions of the soft robot. For example, if the soft robot can be made up of 4 different materials and is limited to a 10^3 voxel array, then there are 5000 possible soft robots and in turn 5000 different genomes. The problem with this type of encoding is that it can take longer for the solution space to be explored and may not even be able to converge to a solution, which becomes even more prominent as the resolution is increased. If the dimensions are doubled, then there are 40 thousand possible solutions. As can be seen, this is not a feasible encoding as the resolution of the soft robot increases [20].

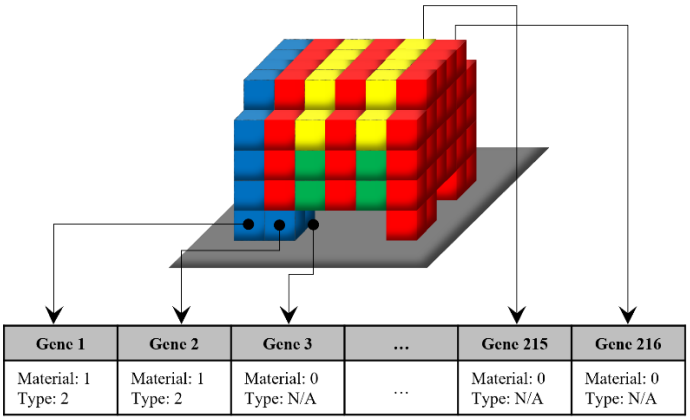


Figure 2: Example of a direct encoding for a soft robot

2.2.2 Directed Graphs

Early evolutionary robotics made use of directed graphs to represent phenotypes of rigid robots. An example of this type of encoding was used by Karl Sims. In Sims' work [18], [21], the graph begins

with a root node, and child nodes can be connected to the root node to add new parts to the core body. Nodes have multiple connections to child nodes to duplicate child structures, and nodes can be connected to themselves to create repetition of their own structure. Examples can be seen in Figure 3. Each node contains information to describe the part as well as the connections between it and other parts. These include dimensions of the part, the degrees of freedom between itself and its parent node, a recursive limit and internal neurons. The recursive limit controls the number of repeated structures that can occur when a recursive connection is present. Internal neurons consist of a function that receives inputs from sensors and then outputs to effectors, acting as a simple brain of the part [18].

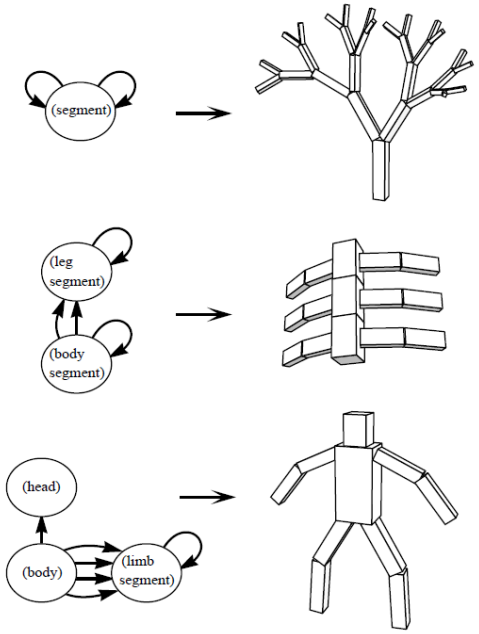


Figure 3: Directed Graph used to encode rigid structures. [18]

2.2.3 Gene Regulatory Networks

As described previously, the natural world encodes trillions of complex structures with only 30 thousand genes. The efficiency of generating trillions of structures from a relatively small set of genes is brought about through developmental biology. To achieve a similarly efficient encoding in evolutionary computation it makes sense to replicate this process artificially, and this is exactly what Gene Regulatory Networks (GRNs) aim to do. Therefore, before we can understand GRNs, it is important to first understand the core mechanisms behind developmental biology [2].

Developmental biology is the study of the process by which organisms develop from a single cell into complex multicellular organisms made up of millions of different types of cells. There are three key mechanisms behind developmental biology – mitosis, cell differentiation, and apoptosis. Mitosis, or cell division, is controlled through hormones and is limited by the presence of neighbouring cells. Due to these mechanisms, some cells continue to divide, such as erythrocytes (red blood cells), whereas

others do not such as neurons. Similarly, different types of cells contain different structures and perform different processes, despite all originating from a single cell. This process is called cell differentiation. Lastly, apoptosis refers to cell death and is a key part in protecting cells from infection but is also a key part in forming the shape of the human body and all its structures within. Apoptosis during the development of an organism shapes many different structures such as joints, hands, and feet and even the nervous system where an estimated seventy percent of the original cells die. All three of these processes are controlled by the presence of specific proteins, which are produced by specific genes. All cells in an organism contain the entire genome for the organism, so how is the protein production only limited to what is required by the cell. This is controlled through the epigenome, which determines which genes should be activated and inhibited, to ensure only the required proteins are produced. Gene expression depends on two components, cis-regulators and transcription factors. Cis-regulators form part of the genome and transcription factors bind to cis-regulators to activate their corresponding genes. Thus, the activation of genes depends primarily on the affinity of the transcription factors and the cis-regulators in the genome, and the concentration of transcription factors inside the cell. This entire process dictates the shape and structure of the entire organism during development as well as the processes and functions of each cell during an organism's life [13, 14].

Gene Regulatory Networks are designed to simulate these developmental processes to achieve the same efficiency in converting simple genomes into complex structures. There are many different implementations of GRNs with varying degrees of accuracy to the real process. However, the most promising implementation in the field of evolutionary robotics is the implementation by Joachimczak et al [13, 15]. The development of a morphology, which includes the cell division, differentiation and cell death of cells is controlled by the GRN. The GRN consists of regulatory units, where each regulatory unit contains, product genes, promoter genes and special genes. Each gene contains a number representing its type of gene, its sign, and a set of N real numbers, which are used to calculate their affinity. Promoter genes determine the activation or inhibition of regulatory units and can either be an additive promoter or multiplicative promoter. Product genes create products that interact with promoter genes to determine gene expression. Lastly, special genes handle the external inputs and outputs of the GRN. A visual representation is shown in Figure 4 [22].

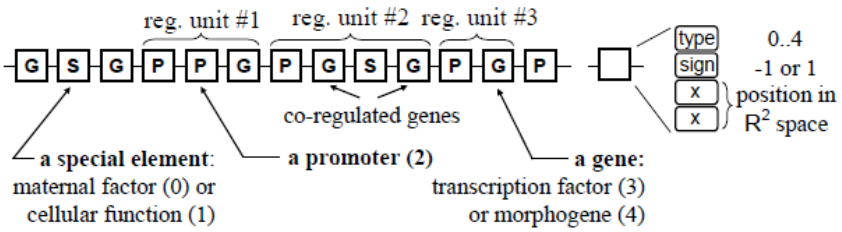


Figure 4: The genome structure used for in GRNs. [2]

The activation of regulatory units is determined by the affinity between the promoters and the products, which can be internal (produced from product genes in other regulatory networks) or externally produced by special genes from neighbouring cells. The affinity between each product and promoter is calculated as a function of the Euclidian distance between the sets of numbers that make them up. In this way connections are formed between regulatory units, giving the genome a network structure. If the Euclidian distance is below a certain threshold the connection is removed to prevent a fully connected network, thus minimising the complexity of the computations required. The activation of each promoter is calculated as the sum of the product concentrations multiplied by their affinities. Thereafter, the activations of the additive promoters are summed and multiplied by multiplicative promoters. Finally, the activation of the entire regulatory unit is determined by inputting the resultant activation into a sigmoid function. This result dictates the production rate of the products from the regulatory unit, resulting in product concentrations changing in time. Multiplicative promoters provide a simple way to inhibit a regulatory unit with a single mutation, as opposed to having to mutate each additive promoter [14, 13, 16].

Special genes encode for the actions taken by the cell, namely, whether it should divide, die, and rotate its orientation (changing the direction of its next division). Additionally, special genes can also be used to dictate the cell type, such as the elastic properties of the cell. Cell division, cell death, and the cell type are determined if the activation of its regulatory unit exceeds a prespecified threshold. To sum up, each regulatory can be considered a node in the network, with product genes and promoter genes forming connections between different nodes, where their weights depend on the affinity and product concentrations, and activation of special genes dictates the processes that occur during the development, as well the cell type. It can be seen how this implementation can be used to evolve a shape of a robot if cell types include cells that can actuate and is exactly what was done in M. Joachimczak et al. [25] to produce complex virtual creatures. Since, the genome has a network structure, a modified version of NEAT can be used to optimise the GRNs and in turn the virtual creatures created [2].

The main drawback to GRNs despite their success and improvements, a lot of computation is required during the development processes in addition to the evaluation of each virtual robot. This limits the number of generations used to evolve the virtual robots, particularly, if the implementation is performed in three dimensions and realistic environments.

2.2.4 Compositional Pattern Producing Networks

Compositional Pattern Producing Networks (CPPNs) provide an alternative way to produce similar shapes and patterns achieved through development without having to simulate the developmental processes. As discussed in Section 2.2.3, morphogens produced by certain genes can be passed from one cell to neighbouring cells, this form of communication provides a way of cells knowing its position

relative to other cells. This positional information is a vital part of generating the shape of an organism through apoptosis and in differentiating cells based on what is required at that position [22].

Similarly, neural networks can produce shapes in 3-D space given positional coordinates as inputs. Additionally, to generate the shapes seen in the natural world such as symmetry and repetition, the activation functions within the nodes of a neural network could be modified to more easily produce the shapes seen in nature. These networks are called Compositional Pattern Producing Networks [26]. CPPNs not only generate the shape of the object but can differentiate a robot into different parts by utilising multiple outputs of the network. In previous work, this has been implemented by using the first output to dictate whether any material is present at the inputted location (if the value exceeds a prespecified threshold). Thereafter, the maximum of the remaining outputs is used to determine which material is used, such as soft, rigid, or actuating. Lastly, since CPPNs have the same structure as ANNs they can be evolved using NEAT [1].

The main limitation of most previous work [9, 11, 18] using CPPNs is that positional inputs are constrained to a three-dimensional grid. This limits the shape of the resultant robots, compared to what can be achieved by GRNs, which grow soft robots from a single cell to form any shape. However, this limitation is only due to the methods used by previous work and not CPPNs themselves, thus the method could be modified to grow robots from a single cell using CPPNs in a similar manner. This type of implementation was shown to be possible by Auerbach et al. [28], where virtual rigid robots were grown from a single part.

2.3 Search Algorithms

In section 2.2, the importance of genetic encoding was discussed to minimise the search space and accurately represent solutions such that optimal solutions can be found efficiently. However, of equal importance is how the solution space is explored. Although evolutionary computation is used to search the solution space, only solutions with desirable characteristics and performance should be allowed to reproduce successive generations. This aspect of evolution, the search and selection algorithm, is what drives the evolution towards a certain objective, and is the focus of this chapter.

2.3.1 Fitness Search

Fitness search is the simplest form of search algorithms. In fitness search, genomes are selected based on their fitness, where the fitness directly relates to the objective. For example, when evolving soft robots to move on land, a simple fitness metric can be defined as the distance travelled by the soft robot in a certain amount of time. The probability of a soft robot being selected to reproduce is proportional to this fitness, resulting in the evolution maximising the distance travelled by soft robots over time. Although a simple metric was used, fitness can be used to drive the evolution towards robots with any behaviour and/or characteristics. Particularly, penalty functions have been used to drive evolution

towards robots with good performance and away from undesirable characteristics. Penalty functions are simple factors that are multiplied by the fitness value to penalise the soft robot for certain unwanted behaviours or features. Some penalty functions which have been used in evolving robots include the amount of energy used by the robot, the size of the robot, etc [9,11, 13].

2.3.2 Novelty Search

Fitness search performs extremely well when there are a limited number of optimal solutions and when the solutions are easy to find, but when the search space is increased and the solutions are more challenging to find, fitness search algorithms tend to get stuck in local minima. To overcome this drawback, the evolution needs to search for completely different solutions, and this is where novelty search comes in.

Novelty search algorithms focus on driving the evolution towards unique solutions as opposed to solutions that just perform well. This method has been shown to significantly improve the solution space explored. Novelty search is implemented in a similar way to fitness search except the probability of a solution being selected is proportional to its novelty. The novelty of a solution is determined using a user-defined function, which represents how different the robot is from other robots in terms of its characteristics or its behaviour [23]. An example of a novelty metric used for evolving robots, is the ratio between a robot's height and its length, resulting in the evolution generating robots of varying sizes and proportions. Unfortunately, since the evolution is driven towards diversity with no knowledge of the objective, the evolution will explore the solution space more effectively but will struggle to find a solution to the problem. This drawback can be overcome by simplifying using a novelty metric, which incorporates the objective within the function [29]. A perfect example of this type of metric is defining the novelty of a robot by comparing how different its end position is to that of previously generated robots [13, 11]. This results in the evolution being driven towards more unique end positions, which will inherently drive the evolution further and further away from the starting point. Novelty search which has the objective built-in to the measure of novelty has been shown to not only improve the solution space explored but is also more effective at finding solutions compared to fitness search techniques [11, 15, 20]. This shows that novelty search algorithms can search beyond local minima, unlike fitness search techniques.

2.3.3 Quality-Diversity Algorithms

It is clear, that both the performance and the diversity of solutions are important factors, which affect the success of evolutionary computation and robotics. However, the objective cannot always be easily incorporated into the novelty metric as discussed in Section 2.3.2. Quality-diversity algorithms are methods of searching the solution space during evolutionary computation by encouraging both the performance of solutions as well as the novelty of the solutions. There are many different proposed methods, which will be discussed [29].

Novelty Search with Local Competition

A simple way to modify the novelty search algorithm described in Section 2.3.2. is to also compare the fitness of solutions when comparing the novelty of solutions. The selection of solutions can be done based on the Pareto rank of each solution, as implemented in NSGA-II multi-objective algorithm. In this way, the evolution is driven to explore the solution space whilst optimising the solutions within each niche of the solution space [20, 21].

Multi-Dimensional Archive of Phenotypic Elites (MAP-Elites)

MAP-Elites approaches quality-diversity search similarly but allows for the niches of the solution space to be explicitly defined. In MAP-Elites, the solution space is defined by specific physical and behavioural characteristics. These characteristics are chosen based on how the user would like the solutions to differ. For example, in evolving robots, this may be the height and length of the robot. Using these characteristics, the solution space may be subdivided into niches of the solution space, where only the fittest solution within each niche is stored throughout the evolution. The population only consists of these solutions, and selection is done at random with equal probability to produce the next generation. These new solutions are evaluated and placed in their corresponding niche. If a solution occupies a new niche or has a greater fitness value than the current solution in its niche, then the solution is stored and added to the population. MAP-Elites has been shown to outperform fitness search in several tasks and is excellent at generating multiple different solutions to solve the problem [22, 23, 24]. MAP-Elites has been used to evolve various control policies which allowed for the control of a robot despite damage to one or more of its legs. This was accomplished since evolution was driven to evolve controllers using different gaits, where some gaits did not require the use of all legs [34]. Some of the limitations of MAP-Elites encourage exploration, making the search slower than that of novelty search. Additionally, since only one solution is stored within each niche, the search is likely to get stuck within a local minimum within each niche [29]. Some modifications have been introduced to overcome these limitations, but do not significantly improve the method beyond what can be achieved through novelty search with local competition [29].

One of the main drawbacks of both quality-diversity algorithms and novelty search is that the niches or novelty metrics are user-defined. This creates a bias towards generating diversity in terms of some characteristics and not others. Even though exploration of the solution space in terms of these other characteristics may be more beneficial.

2.4 Soft Robots

2.4.1 Introduction

No matter how much research and development is made in the design and control of traditional, rigid robots, there will always be limitations to what they can accomplish. Rigid robots can perform repetitive

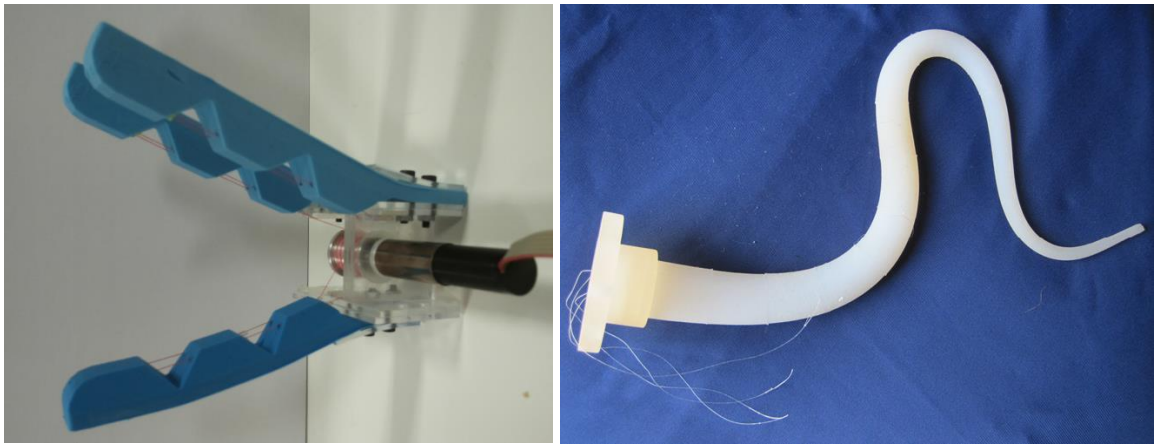
tasks with phenomenal accuracy and precision. However, in turn, when there is uncertainty in the environment or the task changes, then rigid robots underperform. In addition, traditional robots can potentially damage not only the environment but also humans and animals that may interact with, due to the rigidity and lack of compliance. In contrast, soft robots are inherently safer and can handle variations in both the environment and the task, as they are mostly constructed from compliant and flexible materials. These are extremely desirable characteristics of robots in medical applications, applications in nature, handling of delicate objects, and in any applications where robots may interact with people. Soft robots are also inexpensive to manufacture and have the potential of performing significantly more complex motions. A perfect example of the complex motions and tasks that can be potentially achieved by soft robots, is the octopus and its arms. The octopus is entirely soft except for its beak and has been shown to perform a wide range of complex tasks. Its dexterity and motions are achieved through distributed sensing and actuation, something which has not been achieved using rigid materials but has almost been accomplished in the field of soft robotics [35].

2.4.2 Types of Soft Robotic Actuators

The dexterity and degrees of freedom that soft, flexible materials could provide soft robots with the ability to perform more complex movements, hence be able to complete more complex tasks. However, to achieve this level of dexterity, the actuation needs to be distributed throughout the soft material. There has been a wide range of designs that have been developed and will be the focus of this section.

Cable-driven Actuation

In traditional robotics, cables connected to servo motors are used to actuate hinge joints when there is limited space for the servo motors to actuate the joint directly. This is done by threading a cable through a link near the servo motor and attaching it to another link on the other end of a joint. Additionally, when there are multiple joints and links in series, a single cable can be used to simultaneously actuate all of them as shown in Figure 5a. Flexible materials can be thought of as a continuous structure of tiny links and joints, thus the same design and principle can be applied to soft robotics. Using multiple cables, Renda et al. showed how cables could be used to control an octopus-like soft robotic arm, which is shown in Figure 5b. Unfortunately, cable-driven actuators have limited applications and provide limited control of more complex robots as they are not effectively distributed within the material itself [36].



(a) Gripper [37] (b) Octopus-inspired soft robotic arm [38]

Figure 5: Cable driven actuators used to control a gripper and an octopus-inspired soft robotic arm.

Pressure-driven Actuation

The simplest method of distributing actuation within the soft material is using fluid pressure. By pumping fluid into cavities or channels within a flexible material, the material will deform in response to the pressure. The direction and amount of deformation depend on the elasticity of the material throughout, so if the material is symmetrical and has uniform elasticity throughout, the material will simply inflate evenly. However, by designing the material/s used and altering the shape of both the cavity and material itself, the deformation can result in a nonuniform inflation as to achieve a desired motion. A simple example is to create bending motion in a long, sealed tube-like structure, by making one side of the tuber stiffer than the other, when internal pressure is applied by pumping air inside the tube, the tube will bend towards the stiffer material. Alternatively, when making the tube, the channel can be made to be off-centre, which similarly makes the tube stiffer on one side than the other, even though the material itself has a uniform elasticity. This dependence of motion on the shape and stiffness of the soft structure allows for a wide range of designs and applications but can also make it more challenging to find the optimal designs for the desired motion [39].

One of the earliest designs of pressure-driven actuators, which has been used in rigid robots is the Pneumatic Artificial Muscle (PAM). The most widely used design of a PAM is the McKibben actuator (Figure 6), which consists of two end caps connected by an inner elastic bladder and an outer mesh sleeve. The inner bladder is inflated, causing the bladder to expand radially, in turn pulling the end caps towards each other. The outer sleeve is a flexible mesh, which has been woven from a strong fibre. The sleeve constrains the inner bladder from inflating excessively, and more importantly, gives the actuator more strength to resist forces acting in opposition to the actuation. The result is an actuator that contracts like a muscle, hence the name, which can support very high loads [40]. There have been many modifications and uses of this type of actuator, including using multiple McKibben actuators connected

radially. A modified version of this design is currently being proposed for non-invasive surgical procedures (Figure 7) [40].

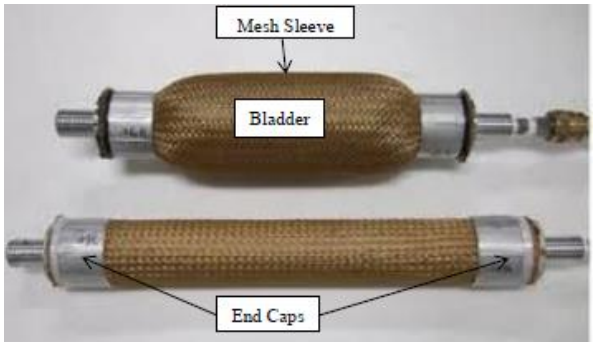


Figure 6: McKibben Actuator [40]

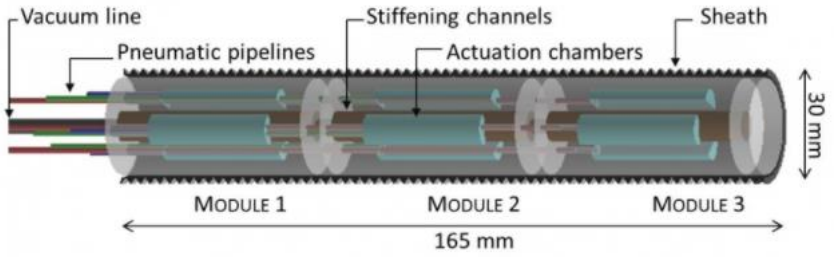


Figure 7: STIFF-FLOP Actuator consisting of multiple McKibben Actuators connected radially [40]

A more recently developed soft robot actuator, which has been specifically designed for soft robots is the Pneumatic Network actuator (PneuNet) [41]. PneuNet actuators consist mostly of a soft elastomer with small cavities along its length and a thinner stiff layer of material of side. When the cavities are inflated, they push against each other extending the actuator along its length, but because of the thin layer of stiff material, resisting the extension, the actuator bends instead as shown in Figure 8. The degree of bending as well as the moment generated can be varied by changing cross-sectional profile. A large cross-sectional area results in large actuating force, and a larger second moment of inertia, which also increases its bending stiffness. These actuators have been widely used in the field of soft robotics including soft robotic grippers, swimming soft robots, and walking soft robots.

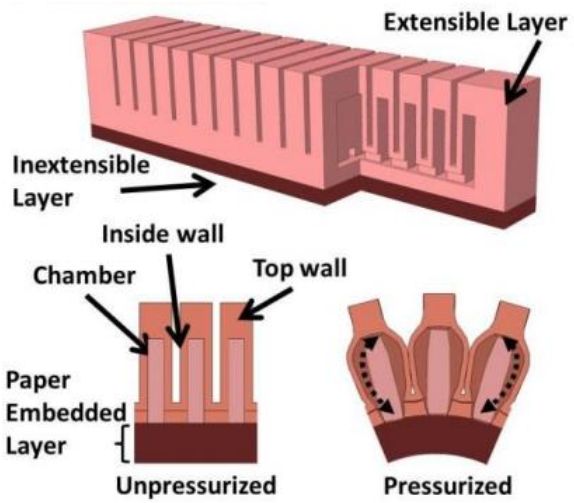


Figure 8: Pneumatic Network (PneuNet) actuator [42]

The pressure in the actuators discussed so far has been created by pumping fluid into the cavities of the soft robot, however, this is not the only way to generate the pressure required for actuation. Alternatively, the pressure can be created through exothermic chemical reactions within the soft robot's cavities. Two chemical reactions have been successfully used to actuate soft robots: combustion of methane and butane, and the decomposition of hydrogen peroxide. The combustion requires a spark in the presence of the fuel and oxygen to initiate the reaction, whereas the decomposition of hydrogen peroxide requires a catalyst such as silver or platinum to begin the reaction. In both cases, the chemical reaction increases temperature, and in turn pressure, thus inflating the chamber within the soft robot. An advantageous property of these types of actuators is that they are self-regulating, in that as the gas moves away from the catalyst or spark and into the cavities, the temperature of the gas begins to cool down. This results in the deflation of the chambers returning the soft robot to its initial shape. In this way, the soft robot can be actuated periodically without the need for external or bulky components [43]. The latter chemical reaction was utilised in the design of the fully soft robot, octobot, which is untethered unlike most soft robots and is only made from soft material incorporating fluidic logic control system [44].

Soft robots do not have to be actuated by increasing the fluid pressure, a negative pressure or vacuum can be introduced into the cavities of elastomer, resulting in buckling and contraction. This can be extremely useful, as vacuum-driven actuators behave like muscles and can actuate similarly high loads. These actuators contract and buckle based on the shape and locations of the cavities, allowing for a wide range of movements [36, 37]. Figure 9 shows how vacuum-driven actuators can be designed to bend, contract linearly, and even generate torque.



(a) (b)

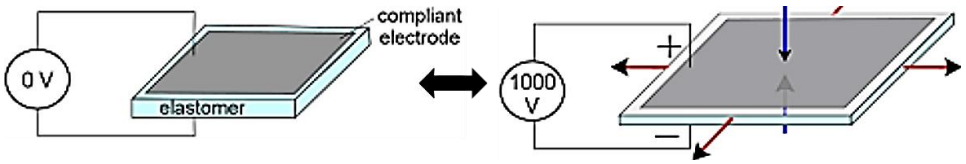
Figure 9: Vaccuum-powered soft actuators used for (a) linear [45] and (b) twisting actuation [46].

Pressure-driven actuators have already been used in the medical field, in underwater and land robots, and for grippers used in manufacturing. However, one disadvantage of these actuators is that most pressure-driven actuators require bulky and rigid components to supply the pressure. Despite this, the dependence of the motion on the shape of both the cavities and the soft robot itself makes current evolutionary robotic techniques such as CPPN-NEAT very suitable for optimising the channels that make up these actuators.

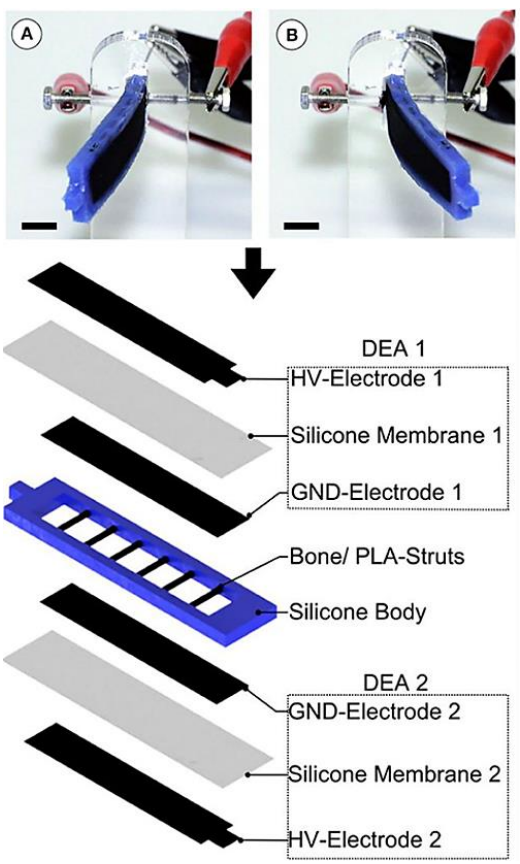
Dielectric Elastomer Actuators

An alternative method for generating motion within a soft material is to take advantage of the Poisson effect. The Poisson effect is the tendency of materials to deform perpendicularly to the load being applied. Dielectric elastomer actuators consist of two flexible electrodes on opposite sides of an elastomer as shown in Figure 10a. The structure resembles a capacitor, but unlike capacitors used in circuits, when the two electrodes are charged, the electrodes compress the flexible elastomer resulting in expansion of the elastomer in perpendicular directions. The flexible electrodes are typically manufactured by setting carbon powder or grease in a matrix of an elastomer giving it both conductive and elastic properties.

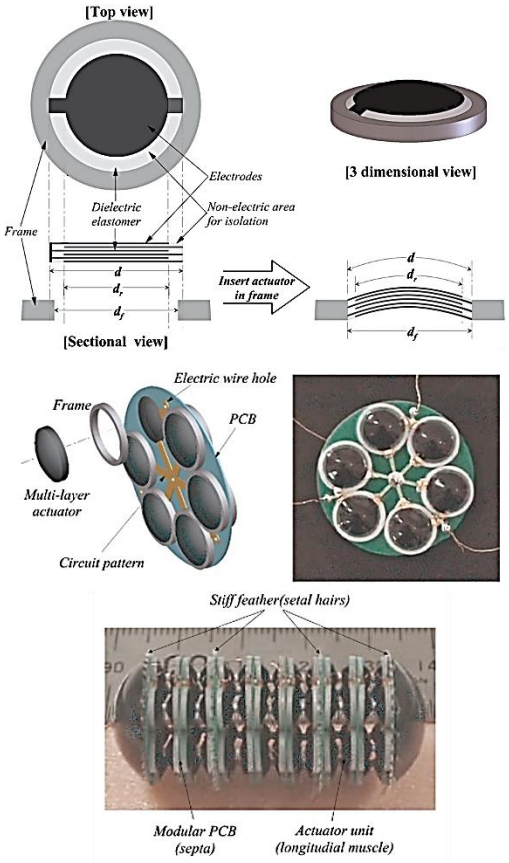
Like pressure-driven actuators, the type of actuation can be altered by changing the shape, stiffness of the actuator. For instance, in M. Franke, et. al's work, two DEAs were stacked on top of one another and inserted in a housing with high stiffness in one direction and low stiffness in the perpendicular direction. The result is a flapping motion by alternating the bending direction through the cyclical charging of the DEAs out of phase with each other [47]. This setup is shown in Figure 10b. A common method of making DEAs is to roll them up forming a cylindrical DEA, which extend when charged. This set-up can be used as artificial muscles, due to their high energy density and power to weight ratio. Lastly, by restricting the expansion of the elastomer, buckling can be intentionally induced to obtain larger deformations of the actuator as can be seen in Figure 10c. In K.Jung, et. al's work, by using many of these buckling DEAs, they were able to design a worm-like robot [48].



(a)



(b)



(c)

Figure 10: Examples of DEAs

A significant disadvantage of DEAs is the high voltage required and is very susceptible to dielectric breakdown and electrical failure. Lots of research is constantly being done to not only reduce the required voltage but also to improve the life of the actuator. One research combined the benefits of both pressure-driven actuators and DEAs to create hydraulically amplified self-healing electrostatic (HASEL) actuators. By replacing the elastomer between the two electrodes with a fluid-filled bladder, compression of the bladder can be used to generate the pressure required for pressure-driven actuation. Additionally, by using a dielectric fluid, the DEA becomes self-healing, since dielectric fluids can restore to their insulating state after dielectric breakdown, unlike solid dielectrics.

Shape Memory Alloys

There are two types of shape-memory materials – shape memory alloys (SMAs) and shape memory polymers (SMPs). As their name suggests, shape memory alloys (SMAs) have the unique property of remembering their original shape despite being plastically deformed. SMAs can take the form of two different crystal structures - martensite and austenite. At low temperatures, SMAs are in the martensite phase, which is capable of large plastic deformations. When the temperature is increased, the SMA transforms to an austenite structure, which generates a restoring force to return the SMA to its original shape before the plastic deformation occurred. By embedding a SMA in a flexible elastomer, the restoring force during SMA's transition can be used as means to actuate the soft material. This temperature change can be induced by introducing an electrical current into the material. The use of SMA's has been mostly limited to small soft robots, where space is limited and high temperatures throughout the SMA can be reached quickly. Unfortunately, SMA's have low efficiency due to heat loss and slow actuation periods, since the SMA needs to cool before a successive actuation cycle can be started. SMAs do provide potential use in combination with other actuators as they can be used to control the stiffness in an actuator. SMAs have been used as the hinge for foldable robots, allowing for the control of folding and shape of the robot. SMPs behave in the same way as SMAs but have a lower actuation strength but with a lower density.

2.5 Simulation Environments for Evolving Soft Robots

Soft robotics is still in the early stages of research and although there has been rapid progress made in the field, there is still limited resources available to simulate soft robots. Most of the simulation software available either focus on the graphics and aesthetics of the soft body material for games and movies or are only capable of simulating the physics of passive soft materials. Soft robots and soft robotic actuators behave in a nonlinear manner, making it difficult to control and design. This makes simulation and modelling of soft robots a vital part of the design process. It allows iterative structural design adjustments and fine-tuning of the controller without manufacturing the soft robot each time. Human-designed soft robots are typically modelled using the finite element method with high resolution to obtain high accuracy and realism. Unfortunately, these simulations take too long to be used in evolving soft robots. The performance of evolutionary computation is highly dependent on the population size of each generation as well as the number of generations, thus requiring an efficient simulation environment whilst maintaining a high level of accuracy.

2.5.1 VoxCad

In Hiller and Lipson's work, they developed VoxCad, which is capable of efficiently designing soft robots made-up of voxels. The soft robot is modelled as a simple mass-spring lattice, but in addition to the mass, the rotational moment of inertia is also stored for each voxel. This provides a more accurate model, where resistance to shear and torsion in addition to extension and compression can also be modelled. VoxCad uses Bernoulli-Euler beam theory to simulate the interaction between adjacent

voxels, limiting the model to small deformations between two voxels. In addition to the physics model used for simulating the interactive forces between the voxels, gravity, collision with both a flat surface and other voxels, and friction are also modelled in VoxCad. The actuation of the voxels was done by modelling the thermal expansion of materials, such that the ambient temperature could be changed to induce volumetric actuation of the voxels. By setting a different thermal expansion coefficient for each material used in the soft robot, different amplitudes of actuation can be achieved.

2.5.2 Simulation Open Framework Architecture (SOFA)

SOFA began with the idea of creating an open-source physics-based simulation software and has developed into a prominent tool to model medical procedures, design medical devices and model soft robotics. Since initial development, SOFA has been made to be modular using plugins, meaning different models and tools can be interchanged and combined with ease. Plugins are constantly being developed and since they are all separated from the core program of SOFA, these additions will not slow down the computation of SOFA. SOFA is written in C++, but scenes can be defined using C++ and XML. Recently though, a new plugin has been developed to allow for the definition of a scene using python 3. This is a significant development as it makes SOFA compatible with a vast number of libraries available for machine learning and data processing, including evolutionary strategies. Additionally, this allows for custom controllers to be created in python, which can interact with a scene during the simulation.

A scene in SOFA is represented using a directed acyclic graph (DAG), as shown in Figure 11. The scene is initialised using a root node, thereafter, subcomponents of the scene and objects in the scene can be added as child nodes. This structure of representation is what gives SOFA its modularity as components and models can be interchanged without changing the entire scene. To ensure that the scene operates correctly despite the modularity, scenes require an animation loop, which manages the order of computation of each subcomponent and model of the scene in each time step. The free animation loop is the type of animation loop used in this research and is used when there are constraints defined in the scene such as attachment between different parts. The free animation loop first builds and solves the system equations ignoring the collisions and constraints, then computes the forces due to any constraints and/or collisions in the scene, which is used to compute corrections to the system, and lastly, the scene is updated. There are many subcomponents and models, which can be used in SOFA, but are beyond the scope of this research, as such only the components and models that are used in this research are described.

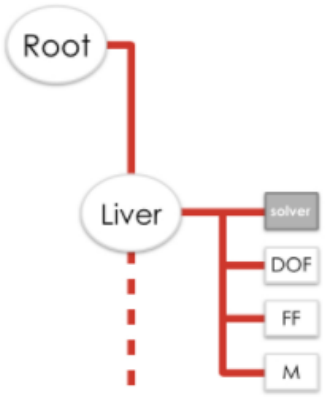


Figure 11: An example of a scene graph in SOFA

There are two core computations performed by the collision pipeline: collision detection and collision response. The collision detection computes whether two objects are colliding or not. To reduce the computational complexity, the detection is divided into broad phase detection and narrow phase detection. Brute force detection is used as the broad phase detection method. This method computes whether the bounding box of two objects intersect or not. For a low number of objects such as a single soft robot, this is a very simple and fast computation. If the bounding boxes are found to intersect, then the collision is passed onto the narrow phase detection component, which for brute force detection components, the bounding volume hierarchy algorithm is used. The function of the narrow phase is to determine specifically what elements of the object's mesh are in contact with each other. These elements are then passed onto the intersection method. The proximity intersection method was used in this research, which computes contact between faces, lines, and points of the mesh if they are below the contact distance threshold. Lastly, this information is passed onto the contact manager, the default was used in this work, which determines the response due to the contact present. In this research, a frictional contact response was required to not only represent a more realistic terrain, but friction is required for locomotion on a terrain. Another important component included in these scenes is a collision group manager, which simply allows for different objects to be grouped preventing the need for collisions to be computed between them, thus reducing computation time. This is required when different parts are attached.

Each object is added to the scene as a child node of the root node. Different models can then be added as children of the object node. This setup allows for different meshes to be used for different physics models. This can aid in reducing computation time, where the accuracy of some of the physics does not matter as much as others. The first sub-components typically defined in the object node is the integration scheme and solver used for the physics computation. The Euler implicit solver was used as the integration scheme and is responsible for building the ordinary differential equations of the system. It does this implicitly, meaning that forces are computed as a function of the state vectors at the next time

step as opposed to the current time step. These equations are then solved using the conjugate gradient method in this research. The next sub-component of an object is the gmsh loader. Although many different mesh loaders are available, gmsh files are sufficiently simple such that an algorithm could be created to automatically generate them. The information loaded is then stored in a topology container, which includes all aspects of the mesh geometry such as the points, edges, faces and elements. The degrees of freedom of an object is defined using the mechanical object subcomponent and store the state of the object at each time step. Although degrees of freedom usually refer to the mechanics of an object such as position, the degrees of freedom can represent other properties that are being simulated such as temperature to model heat transfer. The mass of the object can be defined in the scene using the uniform mass object, which assumes that all vertices share the mass of the object equally. This assumption results in the mass matrix being a simple diagonal matrix, simplifying the computation required to solve the ODEs of the system. Since constraints such as friction and attachment constraints are present in the scene a constraint correction needs to be defined for each object. The component dictates how the constraint correction is applied during the animation loop. Lastly, the physics model that will be used to represent the behaviour of the object, referred to as the type of forcefield in SOFA, is defined. In this research, four different forcefields are used. Hexahedron FEM forcefields are used to model both soft and rigid parts. The elastomer skin is modelled using a triangular FEM forcefield, but the skin would have some thickness to it, a triangular bending forcefield is used to model the resistance the skin would have against bending. Lastly, the pressure inside the actuator is modelled using a surface pressure forcefield. Other useful components, which were used in this research is the topology modifiers and topological mappings, which provided an easy way to convert volumetric meshes to surface meshes to be used as the skin.

In addition to the surface meshes being used to model the actuators, surface meshes were used to model the collision and visual model when needed. A triangle collision model and an OpenGL model was added to each component to model the interactions of objects and visualise the scene, respectively. Lastly, where required attachment constraints were added to the root node. Attachment constraints project the states of each object onto each other forming a geometric connection.

2.6 Domain Randomisation

Domain randomisation is the technique of introducing randomisation in a virtual domain to better represent the noise and variations experienced in real-life scenarios. This method was first introduced by the works of Sadeghi et al. [49] and Tobin et al. [50]. In both papers, deep reinforcement learning is used to train a deep neural network on only virtually generated images. In Sadeghi et al.'s study the deep neural network was used for collision avoidance and in Tobin et al.'s study it was used for object recognition. In both studies, domain randomisation significantly improved the performance of the networks at their tasks. In 2018, OpenAI took it a step further and used domain randomisation to train

a robotic hand completely in a virtual environment to reorientate a cube such that the specified side faces up [14]. Despite only being trained in a virtual environment, the robot controller was able to reorient the cube a median of 13 times. Although this is significantly less than what is achieved in simulations, it still outperforms what is achieved without domain randomisation. OpenAI further optimised their domain randomisation algorithm, which outperformed their previous attempt of reorienting a cube but also used it to train the robotic hand to solve a Rubik's cube.

Due to some similarities between reinforcement learning and evolutionary strategies, it would be expected that domain randomisation could also improve solutions obtained through evolution. Additionally, changes to the environment in the natural world require species to be sufficiently diverse to survive. Similarly, domain randomisation could potentially ensure diversity during the evolutionary process, reducing the susceptibility to get stuck in local maxima and increasing exploration of the solution space.

2.7 Related Work

2.7.1 Evolving Rigid Robots in Silico

Although this work focuses on evolving soft robots, lots of development in the techniques used for evolutionary robotics have come from evolving rigid robots, thus they must be discussed.

Evolving Virtual Creatures

Sim's work is the first work to show how evolutionary computation could be used to evolve not only the controller of a virtual creature but also its morphology [18]. In Sims' work, creatures were created using a directed graph, with each node representing a rigid part and each connection representing a joint between two parts as shown in Figure 2 in Section 2.2.2.

The graph starts at a root node and new nodes and connections are added during the evolution. This method allows for fractal-like structures to be created through recurrent connections and duplicate structures using multiple connections between the same two nodes. Inside each node, the dimensions, joint-type, recursive limit, a list of connections, and neurons for the control of the node are stored. Similarly, each connection stores the position, orientation, scale, and reflection of the child component relative to its parent component. The controller of the creature consists of sensors, the neurons inside each node, and effectors. The sensors could include joint angles, contacts, and the direction a light source. Each neuron contains a function from a variety of available functions, allowing for a variety of computations in response to an input. The effectors are simply the actuators at each joint, where each actuator only controls one degree of freedom at a joint. Connections are formed between sensors, neurons, and effectors, forming a network-like structure, resulting in complex, but effective locomotion of virtual creatures [18].

Binary parameters inside nodes and connections are mutated by simply changing the bit from 1 to 0 or vice versa. Real number parameters are mutated by adding a random number following a normal distribution around the original value. Other parameters such as joint-types and neuron functions are mutated by randomly selecting another joint-type or function from their respective list of possibilities. New nodes are randomly added but have no effect unless a connection is also added between the new node and another. As stated, connections have a random chance of being added, but existing connections also have a probability of being removed throughout the evolution. Reproduction was performed as shown in Figure 12, either crossing-over aligned nodes and connections or by grafting the one onto the other. This research successively showed that evolutionary computation could be used to evolve virtual creatures by successively evolving creatures that were capable of walking and swimming. In later works, Sims made further progress by exploring different methods of selection and applied them to other tasks such as grabbing an object and tracking a target [18].

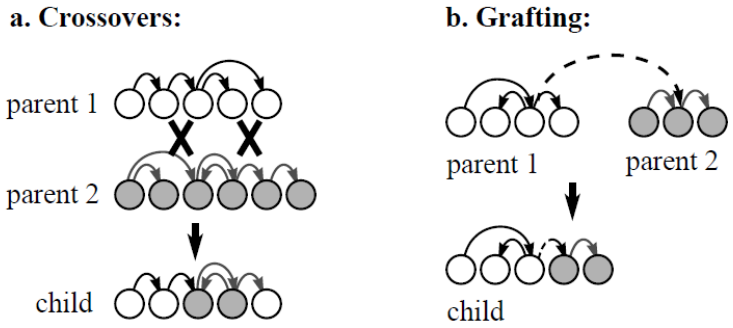


Figure 12: Method of reproduction used to combine two directed graphs in Sims' work. [18]

Since evolved robots were made from rigid parts, morphologies were limited to only combinations of the rigid parts available, which in this work was rectangular prisms. Since the focus of the research was on simply showing the working principle of evolving virtual creatures, the creatures lacked real applicability. This is primarily due to the joints and actuation being discretised. Additionally, the methodology used for the evolution lacked important features, which have been proven to improve evolution such as reproduction using historical markings and speciation. Some of evolved robots are shown in Figure 13.

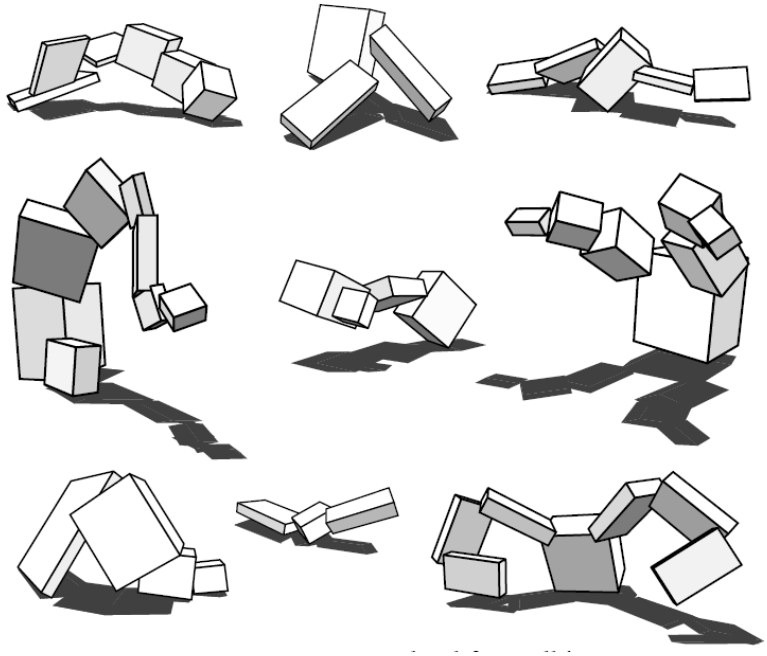


Figure 13: Top-performing Robots Evolved in Sims' Work. [18]

Evolving Complete Robots with CPPN-NEAT: The Utility of Recurrent Connections

In this work, the researchers proposed a method of growing both the morphology and the control policy using CPPN-NEAT. The focus of this study was to show how recurrent connections improved the performance of the evolved robots and hypotheses were made as to why this was the case. Concerning this research, only the methods used for the evolution and properties of the evolved morphology of the robots will be discussed [28].

As stated, CPPN-NEAT was used for the evolution and development of robots. One unique method used in this work in comparison to other's research using CPPN-NEAT is that the robot grows from a central root as opposed to using a predefined grid. The development of a robot begins as a rigid sphere with a predefined radius. Equally spaced points are then cast onto the surface of the sphere and the position of each point is inputted into the CPPN to determine if another sphere should be added at each point. The first output of the CPPN is referred to as the concentration. If the concentration exceeds a predefined threshold, a new sphere might be added provided that no other sphere has already been added at this location and that it does interfere with existing spheres. Excess concentration above the threshold determines the density of the sphere starting from zero. A second output of the CPPN is the scaling factor used to increase or decrease the new sphere based on its parent components size. This allows for differing resolutions and details of the soft robot. If a sphere is added, the third output then determines the type of joint the child sphere has with its parent. If the output is above a predefined threshold, then the joint consists of a single degree of freedom rotational joint where excess above the threshold determines the range of motion. This means that if the threshold is not met, the range of motion of the

joint is zero, thus a rigid joint. The remaining outputs of the CPPN relate to the direction of the joint and the control of the joint, which is beyond the scope of this research [28].

With the aid of new advancements in evolutionary strategies for evolving robots, this research improved significantly on Sims' work. As with Sims' work, the morphologies are still limited by the rigid components, but in this case, a sphere instead of a rectangular prism. Since the joints are limited to a single degree of freedom and parts overlap, meaning joints are not located at a discrete point, a real robot could theoretically be made using this work, but has not been proven. Some of the results from this work is shown in Figure 14.

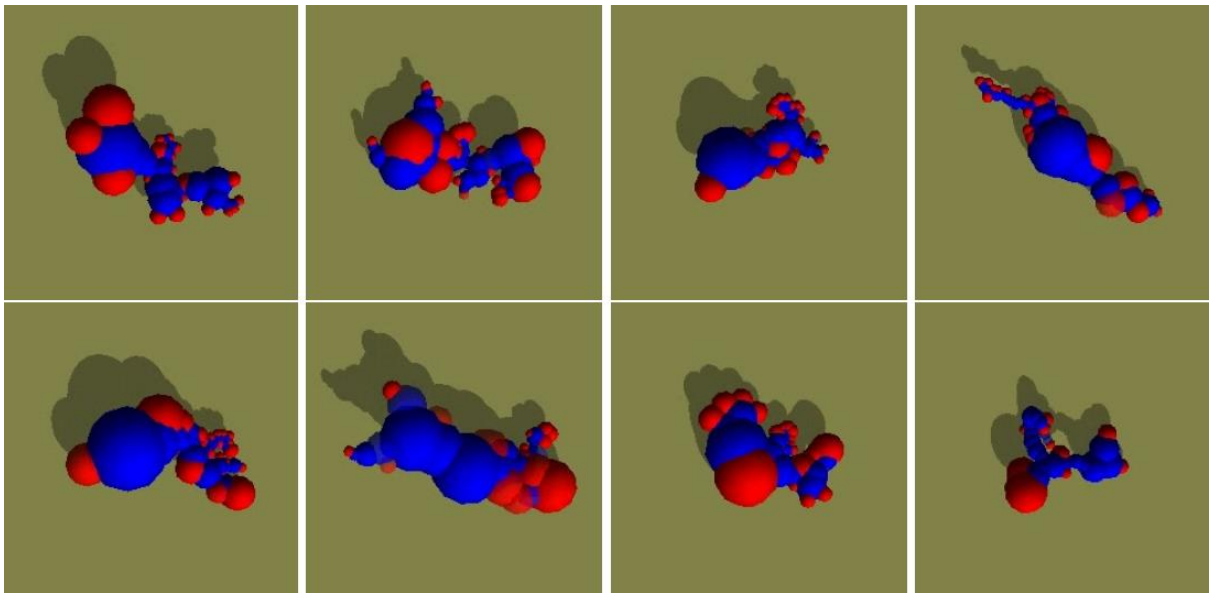


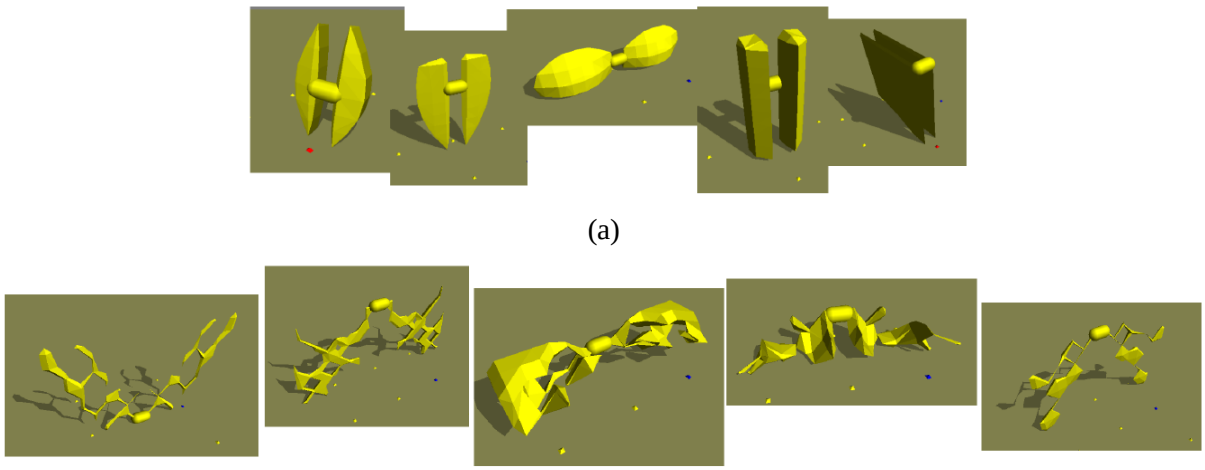
Figure 14: The top-performing robots evolved in Auerback and Bongard's work. [28]

On the Relationship between Environmental Complexity and Morphological Complexity in Evolved Robots

The objective of this research was to analyse how increasing the complexity of the environment affects the complexity of the evolved robots. In this research, CPPN-NEAT was also used to evolve the robot's morphology, but the authors chose not to develop the robot from a central component as they did in their previous work. This decision was made as the authors stated that the robots from the previous work lacked the symmetry and repetition expected from CPPN-NEAT [51].

The authors chose to use a three-dimensional grid of voxels as the inputs to the evolved CPPNs. The significant difference between this research and others that use this same input method is that after the shape is generated from the CPPN, the Marching Cubes algorithm is used to convert the voxel-based mesh into a triangular surface mesh. This provides a smoother and more realistic shape and better represents the output pattern from the CPPN. To focus on the objective of the research, the morphology and control were simplified. This was done by taking the generated shape from the CPPN, making a

reflected copy of it, and attaching these two shapes to the central capsule using two degrees of freedom joints. This method ensured that all robots were symmetrical and that their locomotion was similar. The joints were sinusoidally actuated with amplitudes, ranges of motion, and phase shifts evolved with the CPPN. As with most evolutionary robotics research, the fitness of the robots was equated to the distance travelled by the soft robot during the specified simulation time. To increase the complexity of the environment, equally spaced low friction blocks are added to the environment. Low friction is used such that the robot cannot simply walk on the surface of the blocks but must evolve structures to grip in between the blocks. Using different spacing and heights of the blocks 50 different terrain environments, including a flat terrain with no blocks for comparison, were created. The complexity of the evolved robots was measured in two ways – the ratio between the volume of the robot's bounding box and its actual volume, and a metric defined as the entropy of curvature. Using both metrics, it was observed that more complex environments resulted in significantly more complex shapes being evolved. Some of the results obtained from this is shown in Figure 15 [51].



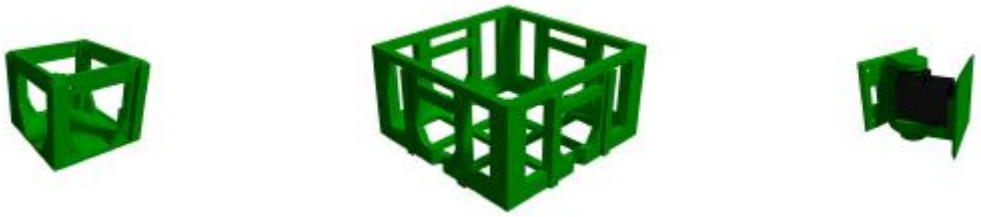
(a)

(b)

Figure 15: The top-performing robots in Auerbach and Bongard's work evolved in (a) a simple environment and (b) a more complex environment. [51]

Lamarckian Evolution of Simulated Modular Robots

In this work, Jelisavcic et al. [52] evolved realistic rigid robots using the RoboGen framework. The RoboGen framework was developed by Auerbach et al. as tool for evolving realistic rigid robots [53]. This was done by designing a set of seven modules that can be easily 3-D printed and connected to create many different robot morphologies. In addition, virtual replicas were made such that these robots can be easily evolved in a virtual environment. The authors in this work chose to only use three of the seven modules to simplify the solution process for easier analysis of the results. These consisted of a central housing from which parts can be attached in four directions, a rigid block and active hinge joint (shown in Figure 16) [52].



(a) (b) (c)

Figure 16: The modules used by the evolution to create robots – (a) a rigid module, (b) the central housing, and (c) an actuated hinge joint. [52]

The objective of this research was to compare Darwin Evolution with Lamarckian Evolution, where learnt behaviours are inherited by offspring to reduce the learning time. Of particular interest to this research, is how the morphologies were encoded and evolved. A direct encoding was used to represent the morphologies of the robots, whereas the controller was encoded using both a central pattern generator (CPG), which depended on the body shape, and CPPNs which was transferred between generations. Thanks to the work of Auerbach et al., this is the first work where robots are evolved in a virtual environment and can be manufactured easily. Some of the evolved robots are shown in Figure 17 [52].

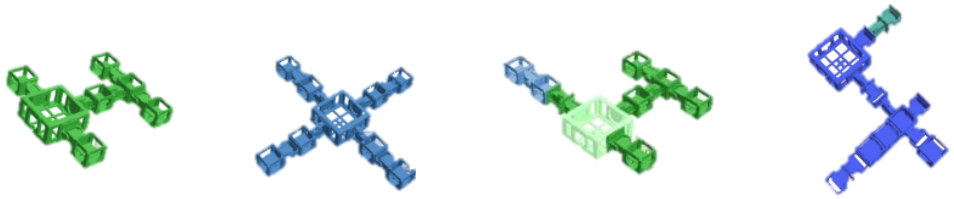


Figure 17: Examples of the evolved modular robots using RoboGen framework [53] and Lamarckian evolution. [52]

2.7.2 Evolving Soft Robots in Silico

Growing and Evolving Soft Robots

This research was one of the earliest attempts to evolve soft robots. The paper focuses on evolving all aspects of soft robotic design – morphology, muscle placement, material properties, and the control of the actuation. To analyse these aspects independently, some of the features of the soft robot were defined initially and kept fixed during the evolution. Only the evolution of the morphology of soft robots will be described here, as it relates to this work [54].

The research developed a L-system, which determines the operation on a tetrahedron's face, to grow a tetrahedral mesh with varying resolution from a single root tetrahedron. Three operations can occur on a tetrahedron's face – the face can be relabelled resulting in different instructions for that new label, it can subdivide thus increasing the resolution, or it can grow by attaching a new tetrahedron to the face.

The soft robots were simulated using Nvidia's PhysX engine and actuation was induced by cyclically changing the stiffness of the tetrahedron's causing distributed deformations as the stiffness changed. An example of an evolved soft robot obtained in this work is shown in Figure 18 [54].

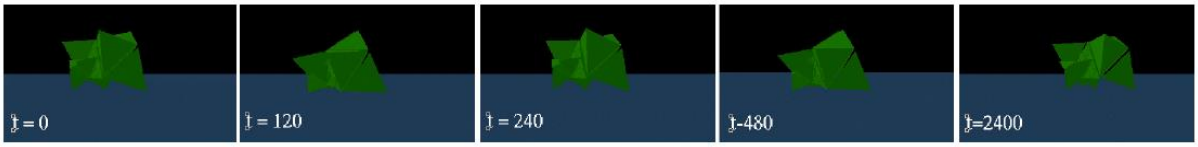


Figure 18: An example of an evolved robot in Reiffel et al.'s work. [54]

One limitation of the work was the encoding used. Firstly, this limited the size and complexity of the soft robots evolved, due to the inefficiency of the encoding in comparison to more modern techniques such as CPPN-NEAT and GRN-NEAT. Secondly, the encoding was applied to equilateral tetrahedrons, which unlike hexahedrons do not fill space, as can be seen in Figure 19 [55]. This means that when growing the soft robot, small gaps in the mesh will form between tetrahedra making it difficult to form large whole structures. Lastly, the actuation method limits the real applicability of the results. Despite this actuation shown to be possible in a vacuum chamber, no further development has been made to realise this type of actuation.

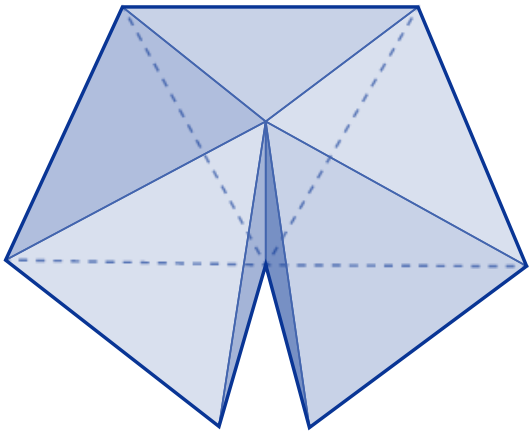


Figure 19: Showing that equilateral tetrahedra cannot fill 3-D space. [55]

Unshackling Evolution – Evolving Soft Robots with Multiple Materials and a Powerful Generative Encoding

Before this research, evolutionary robotics focused on the evolving robots only using rigid components. With the aid of previous research such as the development of CPPN-NEAT [26] and the creation of VoxCad [56], the authors showed that evolving both the morphology and control of soft-bodied robot was feasible. In this work, like others, a three-dimensional grid of coordinates was inputted into evolved CPPNs to generate the shape of the soft robot. The first output of the CPPN determines whether any material is present at the inputted position, whereas the remaining four outputs dictate what type of

material it is. The material can be soft, rigid, or actuating soft material, of which two actuating phases are used. As stated, all simulations are done in VoxCad. The focus of the work was to not only validate that CPPN-NEAT could be used to evolve soft robots, but also to explore the effect of different penalty functions on the distribution of materials in the evolved robots. Penalties were given based on the number of voxels, the number of connections between voxels, and the amount of actuating material. Of significance, without penalties, the evolution performed better, but used more voxels, and penalising the number of actuating voxels resulted in evolved robots using significantly more passive support material. This work is the most similar to the research being performed here, thus will form the basis for comparison. Figure 20 shows some of the evolved soft robots in this work [1].

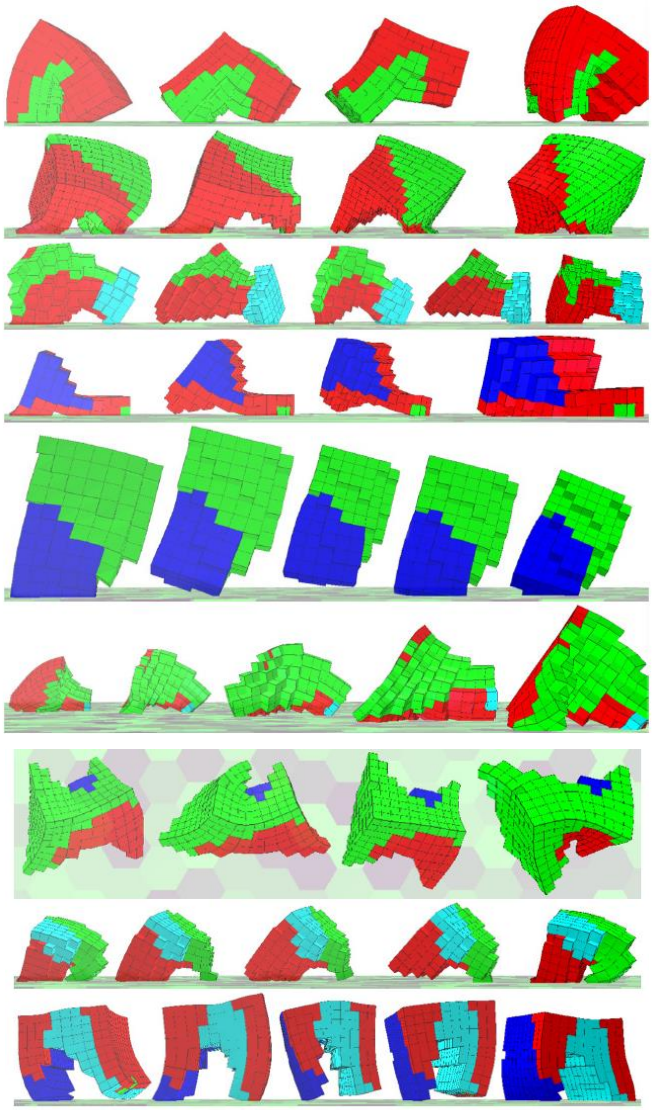


Figure 20: Some of the soft robots evolved in Cheney et al.'s work [1]

Fine-Grained Artificial Development for Body-controller Coevolution of Soft-bodied Animats

Most of the latest research into evolutionary robotics utilises CPPN-NEAT to evolve robot's morphologies and controllers, but this work showed how GRNs can also be used to evolve complex robots. Previous work of the authors had already shown how GRNs can be used to solve the French flag problem and could be used to evolve both the shape and control of a morphology, but due to the amount of computation only simple small creatures were created. By simplifying the computation and approach they succeeded in evolving creatures with over 400 cells, which is a significantly big improvement from their previous work where the size was limited to 32 cells [57]. It is important to note, that despite the simplification, the research was limited to two dimensions because of the computational cost of using GRNs. The robots were modelled as mass-spring systems with internal pressure inside each cell to resist bending and compression. The stiffness of the springs was changed to actuate the robot, where each spring was controlled by the GRN within each cell. This results in non-uniform actuation of the soft-bodied creature, something which is seen in nature but has yet to be achieved in soft robotics. All the above-mentioned limits the real applicability of this approach. Further work has been built on this research, exploring different search techniques, underwater environments and even metamorphous. Some of the top-performing soft robots evolved in this work is shown in Figure 21 [2].

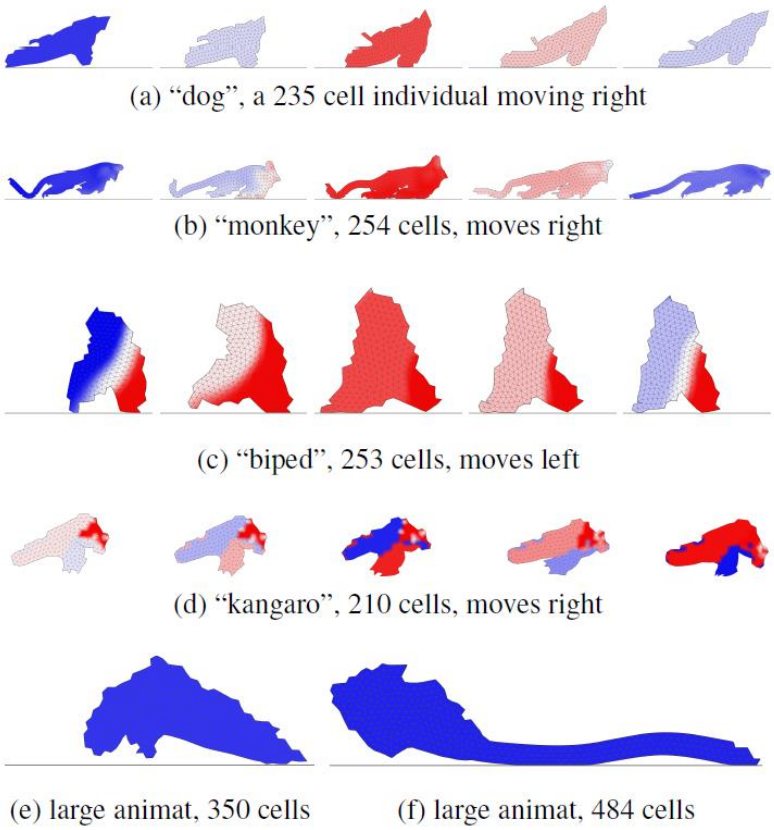


Figure 21: Evolved soft robots using GRN-NEAT. [2]

2.7.3 Evolving Real Robots

A problem with evolutionary computation, as discussed in Section 2.6, is that simulations do not always accurately represent the real world. This results in robots tested in a simulation environment underperforming in the real-world. This is known as the reality-gap. To overcome this issue, the following work attempted to evolve robots in the real-world instead of in a virtual environment.

Morphological Evolution of Physical Robots through Model-Free Phenotype Development

The research aimed at evolving rigid robots using a completely automated evolution, as such the production instructions were encoded into the genome as well as the controller. The automatic evolution was made possible using a robotic manipulator that was capable of producing each robot using the instructions encoded in each robot's genome. Additionally, the manipulator would place the robot a test bed to be automatically evaluated in real life. Evolved robots were composed of a combination of two types of rigid cubes. One of the cubes contained a face that could freely rotate, which was controlled by an internal servomotor. Since the entire production instructions for each robot was instructed in each robot's genome, a simple encoding was required to avoid errors. The authors decided to use a direct encoding as shown in Table 2. Since the evolved robots were only composed of under 5 parts, the scalability of direct encoding was not an issue [58].

Table 2: An example of a directly encoded genome used in this work. [58]

Field	Gene 1	Gene 2	Gene 3
structure	[0°, 0°]	[-90°, 0°]	[-90°, 90°]
Type	2 (active)	1 (passive)	2 (active)
orientation	[90°, 0°]	[0°, 0°]	[0°, -90°]
rel pos	[0, 0]	[0, 0.8]	[0.6, 0.8]
Area	0.5	0.8	0.2
phase shift	4	0	3
amplitude	5	0	5
rot end[a]	1		

Although, this research successively evolved robots that can operate in the real-world, evolving robots in the real-world has severe drawbacks. Firstly, each robot has to be evaluated in real-time in contrast to what can be achieved through parallel computing and optimised simulation environments. Secondly, since each robot has to be manufactured, evolved robots are limited in the number of parts they can be made-up of and in their size. For these reasons, the authors only succeeded in evolving robots 10 generations resulting in robots made-up of no more than 4 parts. A few of the results can be seen in Figure 22.

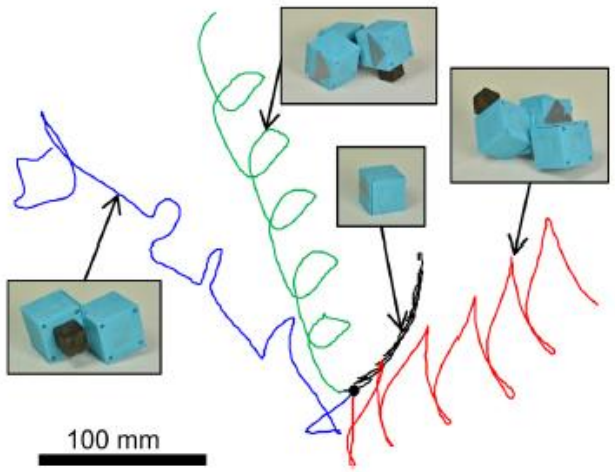


Figure 22: Examples of the achieved robotic behaviours and morphologies evolved in this work. [58]

2.8 Identified Gaps in Literature

Despite numerous successes in evolving soft robots, there has been no attempt at evolving virtual soft robots that can be realistically manufactured, as has been done by Jelisavcic et al. [52] with the use of the RoboGen framework [53]. Additionally, there is a huge mismatch between how virtually evolved soft robots are actuated and the existing soft robotic actuators presented in Section 2.4.2. Lastly, only one of the discussed attempted to close the reality gap by evolving robots in the real world; however, this has major drawbacks as discussed in Section 2.7.3, which limit its scalability and practicality to be used to design soft robots for real-world applications. Alternatively, domain randomisation has been shown to significantly improve the performance of controllers in the real-world when used by reinforcement learning. Domain randomisation has not been utilised in evolutionary computation to the author's knowledge. Lastly, as can be observed by comparing the results obtained in previous research, some of the studies presented limited the evolved morphologies of soft robots to a 3-D grid, substantially reducing the complexity, shape and size of the soft robots that could be produced.

3 RESEARCH SCOPE AND OVERVIEW

3.1 Research Objectives

The core objective of this research is to improve the real applicability of soft evolved using CPPN-NEAT. This objective will be fulfilled by completing the following sub-objectives:

- Modify and build on the methods used in previous works to improve the performance of CPPN-NEAT at evolving soft robots.
- Evolve soft robots in a more realistic simulation environment, Simulation Open Framework Architecture (SOFA), to reduce the reality gap.
- Evolve soft robots that utilise realistic fluidic elastomer actuators.
- Determine the effects of domain randomisation on the performance of CPPN-NEAT and the exploration of solution space.
- Analyse how growing soft robots from central element effects the evolution of soft robots compared to the grid mesh method used in previous works.

3.2 Research Scope and Limitations

The research can be carried by completing the following tasks to meet the objectives:

1. Perform thorough literature review of evolutionary robotics and soft robots.
2. Develop an algorithm to automatically generate mesh files from the outputs of the evolved CPPNs to be imported into SOFA using both a grid of positional inputs and inputs growing outwards from a central element.
3. Set-up a robust simulation scene in SOFA that can consistently simulate all soft robots accurately.
4. Develop a controller that controls not only the actuation during the run but initialises the scene based on the soft robots' properties.
5. Develop an error handling algorithm to exclude soft robots that crash the simulation and soft robots that behave unrealistically.
6. Complete 5 evolutionary runs using different random seeds for 200 generations to evolve soft robots. Repeat this for the following modifications and methods:
    a. Modifications to the NEAT settings.
    b. Modification to the fitness function.
    c. Novelty search.
    d. Fitness search with the inclusion of domain randomisation during the evolution.
    e. Growing the soft robot structure from a central mesh element.
7. Compare all the results and perform a complete analysis on all the methods and modifications used.
8. Present and discuss the results from all analyses and compile all findings.

3.3 Research Contributions

Some minor improvements are made to the current methods of CPPN-NEAT for evolving soft robots in a virtual environment, but the core contributions made by this research to the field are:

1. This is the first research to the author's knowledge to use SOFA to evolve soft robots, providing more realistic results and narrowing the reality gap.
2. This is first research to author's knowledge that realistic fluidic elastomer actuators have been incorporated in the evolution of virtual soft robots.
3. To author's knowledge, this is the first use of domain randomisation not only in evolving soft robots, but in evolutionary computation in general. This research shows the effects of domain randomisation on the performance of CPPN-NEAT and the exploration of the solution space, as well as how domain randomisation can be used to evolve more robust soft robots to aid in closing the reality gap.
4. This research has proposed a method of growing the hexahedron mesh structure of soft robots to improve performance and expand the solution space.

3.4 Dissertation Layout

The remainder of the dissertation is divided into five more chapters consisting of the methods, the set-up, the results and discussion, conclusions, and recommendations for future work. The methods describes the how comparisons and analyses will be performed including defining any performance metrics, figures, and statistical tests that will be used. Following the methods, a detailed description on how the research was set-up to perform the evolutionary runs and tests. This chapter includes the set-up of the simulation environment in SOFA, the set-up of NEAT and the evolutionary settings used, and descriptions of all the algorithms developed during the research. The results are presented and discussed in the following chapter, Chapter 7. The results and discussion are divided into the different analyses, starting with comparisons between the set-up used in this research versus that used in previous work. Following this analysis, the effects of domain randomisation are presented and discussed. Next, the effects of using a growing mesh instead of grid mesh is discussed. Lastly, results obtained using SOFA as the simulation environment is compared to that obtained using VoxCad. The recommendations for future research contains potential improvements to this research based on the results obtained, as well as further research, which was simply beyond the scope of this dissertation.

4 RESEARCH METHODOLOGY

4.1 Introduction

This section provides details on the methodology of the analyses that were performed. Three analyses were taken out during this research. First, domain randomisation is implemented in the evolutionary process and the performance and exploration of the solution space are compared to search methods without it. Secondly, a different mesh generation method is used and compared to that of previous research. Lastly, a comparison is made between the soft robotics simulation software used in this research versus that of previous research.

4.2 Validation of Modifications

Before analyses can be done, it was important to make a comparison between the set-up of this research and previous studies, to show that the modifications were either necessary or do not affect the performance of the evolution. Some of the modifications were necessary to compensate for the increase in computation time required by the more realistic simulation models used.

4.2.1 Modification to NEAT Initialisation Settings

The first modification made was to the evolution parameters. In previous research, the evolution begins by randomly generating fully connected CPPNs with no hidden nodes. However, this already results in complex shapes and soft robot layouts, which goes against one of the core advantages of NEAT – starting simple and evolving to more complex solutions only if necessary. In addition, there is evidence to show that better performance can be achieved by starting the evolution with networks that are not fully connected [59]. This allows for feature selection, as not all inputs are important and necessary to generate an optimal solution. However, starting the evolution with no connections results in generations of trivial soft robots with no complexity. A middle ground was found by generating the CPPNs, where 50% of connections between all the inputs and all the outputs are activated. Due to the increase in computation time, runs had to be limited to 200 generations. This is a significant reduction from the previous work where soft robots were evolved for 1000 generations. To account for the reduction in the number of generations and the simpler initial networks, the mutation probability for connections and nodes to be added was increased to 0.4 and 0.2 from 0.05 and 0.03, respectively, and to be removed was increased to 0.2 and 0.1 from 0.05 and 0.03, respectively. The remainder of the CPPN-NEAT settings were kept the same as that used in previous research. These settings are shown in Table 3.

Table 3: The evolutionary settings used in this research.

Evolution Parameter	**Value**	**Description**
Activation Default	Sigmoid	This is the initial activation function used by all nodes.
Activation Mutation Rate	0.2	The probability that the activation function will change.
Activation Options	Gaussian Identity Sine Sigmoid Absolute	A list of possible activation functions a node can have, which allows the networks to generate more natural shapes.
Bias Initial Mean	0	The initial mean value for mutations to a bias.
Bias Initial Standard Deviation	1	The initial standard deviation for the mutations to a bias.
Bias Maximum	30	Maximum value a bias can have.
Bias Minimum	-30	Minimum value a bias can have.
Bias Mutate Power	0.8	How much a mutation can change the bias.
Bias Mutate Rate	0.8	The probability that a bias will be mutated by adding a random value.
Bias Replace Rate	0.1	The probability that a bias will be replaced with a new random value.
Compatibility Disjoint Coefficient	2	Coefficient factor to weight the contribution of disjoint and excess nodes and connections between two genomes to the genomic distance.
Compatibility Weight Coefficient	1	Coefficient factor to weight the contribution of the difference in connection weights between two genomes to the genomic distance.
Compatibility Threshold	6	The threshold value for the genomic distance between two genomes needs to be surpassed to separate them into two species.
Connection Add Probability	0.4*	The probability that a new connection will be added
Connection Delete Probability	0.2*	The probability that a connection will be removed
Node Add Probability	0.2*	The probability that a new node will be added
Node Delete Probability	0.1*	The probability that a node will be removed
Initial Connection	0.5*	The connectivity of the initially generated network, 1.0 indicates a fully connected network and 0 indicates no connections.
Hidden Nodes	0	The number of hidden nodes in the initially generated networks.
Input Nodes	4	Number of inputs to the CPPNs (x, y, z, r: distance to the centre)
Output Nodes	5	The number of outputs of the CPPNs (Material is present, soft material, rigid material, actuator 1 (ϕ=0°), actuator 2 (ϕ=180°). Material is present at the inputted location if the first output > 0.3, thereafter the

		maximum value of the remaining outputs determines what material it is.
Weight Initial Mean	0	The initial mean value for mutations to a weight.
Weight Initial Standard Deviation	1	The initial standard deviation for the mutations to a weight.
Weight Maximum	30	Maximum value a weight can have.
Weight Minimum	-30	Minimum value the weight can have.
Weight Mutation Power	2*	How much a mutation can change a weight.
Weight Mutation Rate	0.8	The probability that a weight will be mutated by adding a random value.
Weight Replace Rate	0.1	The probability that a weight will be replaced with a new random value.
Maximum Stagnation	15	The number of generations before a species is removed if no improvement is made.
Species Elitism	2	The number of the top-performing species, which cannot be removed.
Elitism	2	The number of top-performing solutions in each species, which are protected, and remain unchanged every generation.
Survival Threshold	0.2	The proportion of solutions in a generation selected to survive and reproduce the next generation.

* Indicate the settings that have been changed from previous research.

Data Analysis

The modifications made only relates to how the complexity and size of the CPPNs change per generation. As stated, this was done to account for the reduction in the number of generations, thus it was important to show that this modification does increase the rate at which the CPPNs grow during the evolution. This was done by defining the solution space as a two-dimensional grid with each element in the grid corresponding to a specific combination of the number of nodes and connections of a CPPN as shown in Figure 23. This solution space was recorded as a heatmap, where each element in the grid stores how many CPPNs during the evolution has the combination specified by the element. Using this definition, the amount of exploration of the solution space in terms of the complexity of the CPPNs was determined by how many elements in the heatmap have a value greater than one. This quantity was compared between the evolution settings from previous research and the modified settings used in this work. The comparison was made by comparing the median exploration metric of five different runs using Wilcoxon's rank sum test.

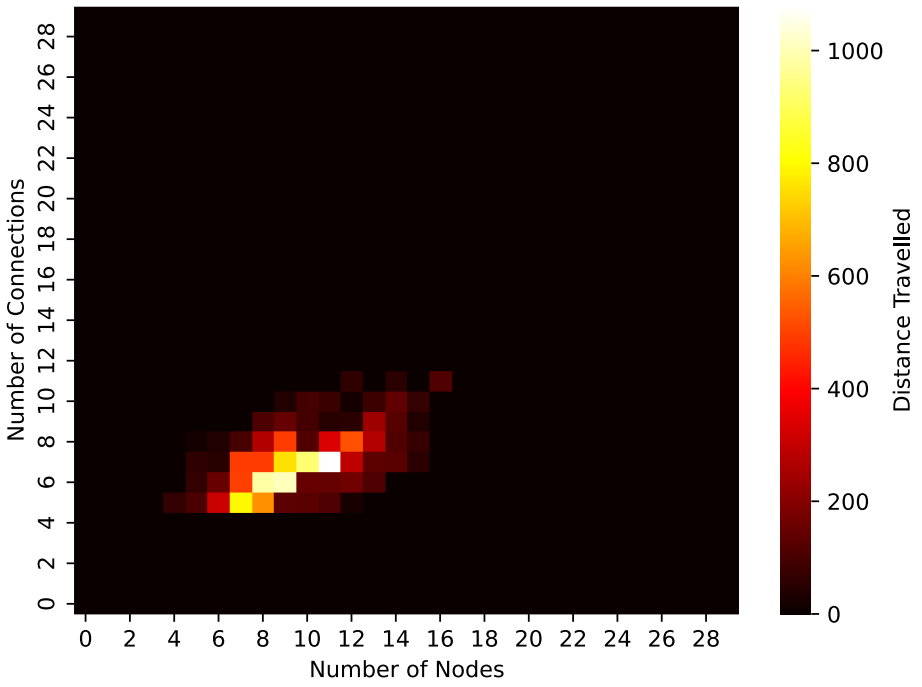


Figure 23: Example of a heatmap showing the exploration of the solution space in terms of CPPN structure.

It is also essential that the increase in the rate at which the CPPNs grow does not affect the ability of the evolution to converge on a specific solution, thus the performance of the two settings was also compared. The performance of an evolutionary run was measured by the distance travelled by the robot with the highest fitness at the end of the evolution. The median distance of five different evolutionary runs was compared using Wilcoxon's rank sum test.

4.2.2 Modified Fitness Function

The fitness function is a crucial aspect that influences the success of evolutionary computation tasks. Evolutionary computation is extremely effective at finding local minima and simple solutions. If a fitness function is not carefully selected, the evolution will exploit any oversights or shortcomings of the fitness function. A good example has been seen in past research in evolving terrestrial robots. When the fitness is simply defined as the distance travelled by the soft robot as it has been in past research, the evolution converges on tall, slim passive structures, which simply fall over resulting in a decent distance being achieved [25]. There are three fitness functions used throughout this research – total distance travelled, the average distance of the two smallest segments, and novelty search.

Total Distance

The first fitness function was used in previous work and was only used as a comparison to the modified fitness function. This fitness function is simply the distance travelled starting from after the initialisation period until 15 actuation cycles have been completed. The fitness is measured using equation 1.

$$f = |\vec{x}_0 - \vec{x}_{15}| \tag{1}$$

One disadvantage of this fitness function is that falling results and rolling results can perform well despite not moving by utilising the actuators. With NEAT being susceptible to getting stuck in local minima this can result in the evolution converging on these simple exploits of the fitness function.

Average Distance of the Two Smallest Segments

Alternatively, in this research, a modified version of the distance function was used to encourage consistent movement as opposed to short, sudden movements. This was done by recording the displacement travelled in three consecutive segments (5 actuation cycles each). The total displacement was recorded to provide the average direction of the soft robot after the run. The distance of each segment in the direction of the final displacement was calculated. Finally, the fitness of the soft robot was equated to the average of the lowest two distances. This fitness function ensures two things, any sudden but inconsistent movement during the run such as falling or rolling will not contribute to the soft robot's fitness, and soft robots, which do not move in a straight line will get a lower fitness value than those that do. This fitness function is described using equations 2 to 4.

The direction of final endpoint:

$$\hat{\boldsymbol{u}}_{\boldsymbol{f}} = \frac{\vec{\boldsymbol{x}}_{\boldsymbol{f}}}{|\vec{\boldsymbol{x}}_{\boldsymbol{f}}|} \tag{2}$$

The distance towards the endpoint in the ith segment:

$$x_i = \vec{\boldsymbol{x}}_{\boldsymbol{i}} \cdot \hat{\boldsymbol{u}}_{\boldsymbol{f}} \tag{3}$$

The fitness of the soft robot is then:

$$fitness = \frac{x_{min} + x_{2nd\ min}}{2} \tag{4}$$

Figure 24 provides examples of how this fitness function is implemented and how it penalises inconsistent motion.

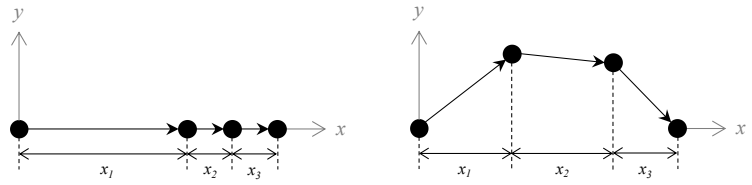


Figure 24: Visualisation of how the modified fitness function is calculated and how it would penalise inconsistent motions.

Novelty Search

Novelty search is used to encourage more exploration of the solution space during the evolution. As stated in Section 2.3.2, novelty search only performs better than fitness search methods when the novelty metric used relates to the objective. In this research, the position of the robot was used as the novelty metric. As with fitness search, novelty search using the end position of the robot can result in the evolution converging on falling and rolling structures with no actuating materials. To avoid this, the novelty search is similarly modified to make the two searches more comparable. The calculation of the position used by the novelty metric is described by equations 5 to 7 and examples are shown in Figure 29.

The direction of final endpoint:

$$\hat{\boldsymbol{u}}_{\boldsymbol{f}} = \frac{\vec{\boldsymbol{x}}_{\boldsymbol{f}}}{|\vec{\boldsymbol{x}}_{\boldsymbol{f}}|} \tag{5}$$

The distance towards the endpoint in the ith segment:

$$\vec{\boldsymbol{r}}_{\boldsymbol{i}} = \vec{\boldsymbol{x}}_{\boldsymbol{i}} \cdot \hat{\boldsymbol{u}}_{\boldsymbol{f}} \cdot \hat{\boldsymbol{u}}_{\boldsymbol{f}} \tag{6}$$

The position used to calculate the novelty:

$$\vec{\boldsymbol{r}}_{\boldsymbol{novelty}} = \frac{\vec{r}_{i_{min}} + \vec{r}_{i_{2nd\,min}}}{2} \tag{7}$$

The only difference between the two methods was that in the fitness search the average of the two smallest distances was used, whereas in this method the average of the two smallest displacements was used. The average Euclidean distance between this displacement and the *k* nearest displacements other soft robots in the current generation and an archive of previous novel solutions is calculated using Equation 8. The archive stores the *n* most unique positions found during the evolution, which is updated every generation. In previous research, it was shown *k=10* and *n=1000* were the optimal parameters for this type of novelty search.

$$novelty = \frac{1}{k}\sum_{i=1}^{k} |\vec{\boldsymbol{r}}_{\boldsymbol{novelty}} - \vec{\boldsymbol{r}}_{\boldsymbol{i}}| \tag{8}$$

It is important to note that quality-diversity algorithms were not used in this research so that purely novelty and purely fitness search can be compared with evolutions using domain randomisation.

Data Analysis

It was hypothesised that the modified fitness function would drive the evolution towards soft robots that move consistently in a straight line. To validate this statement, the locomotion of the best performing soft robots using each fitness function was compared. This was done by recording the speed and path of the soft robots during the simulation. In addition, the median total distance travelled was compared using Wilcoxon's rank sum test to determine whether the new fitness function improved the overall performance or not.

4.3 Effect of Domain Randomisation on Performance and Diversity

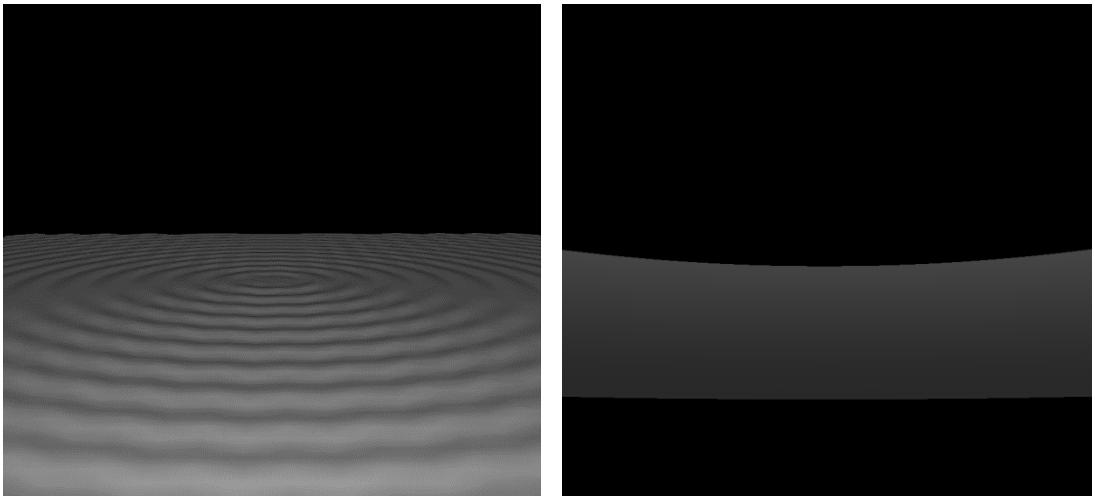
To determine the effect of domain randomisation on the performance of the evolution, the approach was compared against fitness search and novelty search. The performance of the evolution can be quantified by how well the solution space was explored and how well the evolved soft robots performed. The performance of the soft robots was determined in two ways. Firstly, the average total distance of the top-performing soft robots from five different runs for each method was compared using Wilcoxon's two-tailed test. More importantly, domain randomisation was used to evolve soft robots that can handle variations in the environment, thus improving the sim-to-real transfer. To test how well domain randomisation can be used to evolve more robust solutions, the performance of the top-performing soft robots from each method was tested in unseen environments. The unseen environments include three different uneven surfaces and three sloped surfaces of varying degrees of difficulty. The bumpy surfaces are generated by radially adjusting the height of each vertex of the floor mesh (y) according to a sine wave given by equation 9. The frequency of the sine curve (ω) was adjusted to generate three different uneven surfaces.

$$y = \sin \omega r \tag{9}$$

The sloped surfaces were generated by increasing the height of the surface mesh (y) as the distance from the centre increases (r), according to a quadratic surface given by equation 10. To increase the level of difficulty the rate at which the slope gradient increases (m) is increased.

$$y = mr^2 \tag{10}$$

The median total distance achieved by the top-performing soft robots was recorded and compared. More importantly, the performance on the new terrains was also recorded as a percentage of its performance on the flat terrain to quantify how well the performances of the soft robots translates to an unseen environment. Examples of the two of the size terrains are shown in Figure 25.



(a) Bumpy terrain (b) Sloped terrain

Figure 25: Examples of the unseen terrain for testing soft robots' ability to adjust to new environments.

As describe in Section 4.2.1, the solution space can be defined using a grid; however, in this analysis, the structure and behaviour of the soft robots evolved were of interest. Instead of using the number of connections and nodes to define the solution space, metrics describing the structure of the soft robots were used. A soft robot's structure can be described using a variety of metrics, but in this work, the structure of the soft robot was described using the centroid positions of the two actuators and the number of passive support voxels versus the number of actuating voxels. The positions of the actuators were used as it not only describes the structure of the soft robot but also relates to the resulting behaviour due to the actuator placement. The number of passive voxels versus the number of actuating voxels describes both the composition of the soft robot and the overall size of the soft robot. These metrics were used to create heatmaps showing how many different combinations of these features were used. An example is shown in Figure 26.

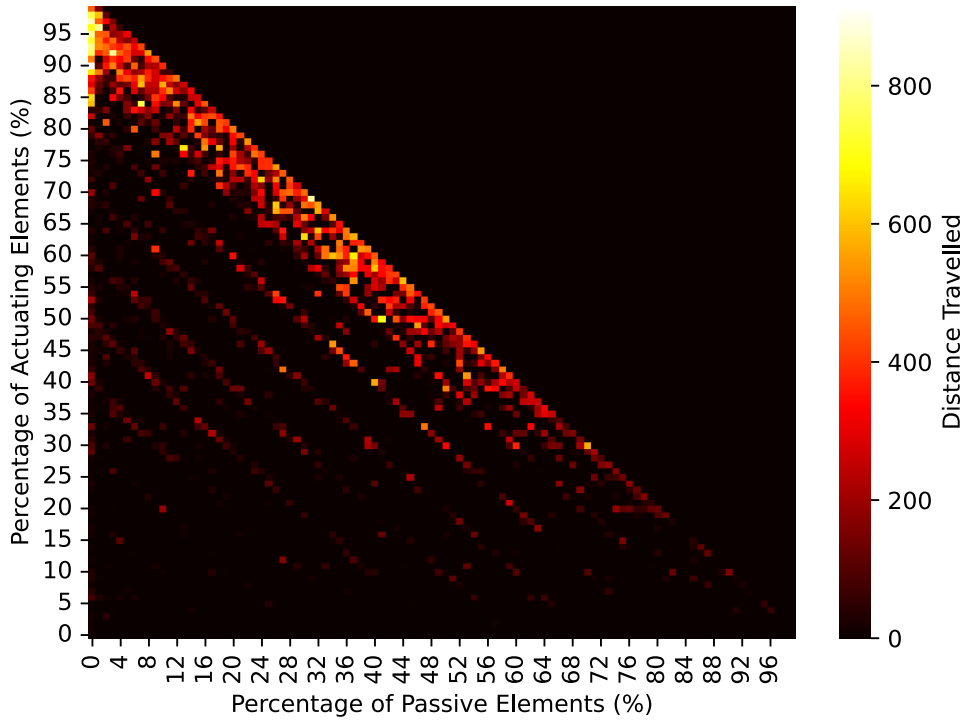


Figure 26: An example of how a heatmap can be used to observe the exploration of the solution space.

4.4 Increasing the Solution Space through Mesh Growth

It was suggested that the 3-D grid method used in previous work to query the CPPN limits the shape and the solution space, and possibly in turn the performance of the evolution. Each method is used for 5 runs using the same random seed in both methods. In the evolution using the grid input method, the grid is limited to a cube grid with a dimension of 150 mm made up of 10 elements each. The evolution using the growing mesh method limits the number of elements to thousand elements of the same size to ensure that the maximum mesh size of both methods is the same. In addition, the length, width, and height of soft robots are not allowed to exceed 500 mm, as excessive collision computation is required. All evolutionary runs are performed for 200 generations with a population size of 64 using the modified fitness function described in Section 4.2 and the evolutionary settings specified in Section 4.1. As with previous analyses, the performances of the two approaches were compared using the median total distance travelled by the top-performing robots and the Wilcoxon's rank sum test.

To determine the effects of the two methods on the solution space and evolution, the length, width, and height of all evolved soft robots was recorded during the evolutionary runs. After all the runs were completed, the distribution of these three attributes was compared to determine the extent by which the grid method constrains the solution space.

4.5 Comparison between VoxCad and SOFA

The primary reason for comparing the two environments is to verify the necessity of a more realistic simulation environment and physics models provided by SOFA. To compare the two environments, qualitative measures and observations were made. No quantitative measures are used, because similarity in the behaviour and motion is more important than the distances achieved in determining if the simplified physics models can effectively model more complex physics.

First, three evolutionary runs were performed in VoxCad to obtain the top performing soft robots found in three different runs. Then the physical designs of the evolved soft robots are imported into a SOFA environment with similar properties. The simulation for each soft robot is run in SOFA and observation is made on the behaviour of the soft robots and similarity between the two simulation environments. It is important to note that although attempts were made to make the environmental properties and material properties as similar as possible some differences remain due to the underlying physics models. For example, since each actuating element in VoxCad actuates independently from each other instead of being combined into a single actuating cavity as in SOFA, each element also has its own mass which is absent inside the cavity in SOFA. Additionally, the elasticity models in the two environments are different, so the elastic constants are difficult to equate. The purpose of this analysis is to determine whether these underlying differences substantially changes macroscopic behaviour of the soft robots in the two environments.

5 EXPERIMENTAL SET-UP

5.1 Overview of the Set-up and Procedure

This section provides the set-up that was used for all experiments performed and includes detailed descriptions of the core algorithms, libraries and simulation software used. Before any analyses could be performed, a base set-up of the evolutionary algorithm and all its subcomponents was required. An overview of the entire evolutionary computation is summarised in Figure 27. The evolutionary process was initiated by generating an initial population of CPPNs. Each CPPN received positional information as inputs and in return, outputted whether the material was present at the specified location and what type of material it should be. The position and corresponding outputs were then used to generate mesh files to be imported into the simulation environment. Inside the simulation environment, the soft robot was simulated for 15 actuation cycles. At the end of the simulation, the results from the simulation were used to evaluate each soft robot using a fitness function. The fitness values of each soft robot were used by NEAT to perform selection and reproduction to create the next generation of CPPNs. This process was repeated for 100-200 generations. The evolution was run in parallel on 64 CPU cores, resulting evolutions ranging from approximately 4 hours and up to 24 hours.

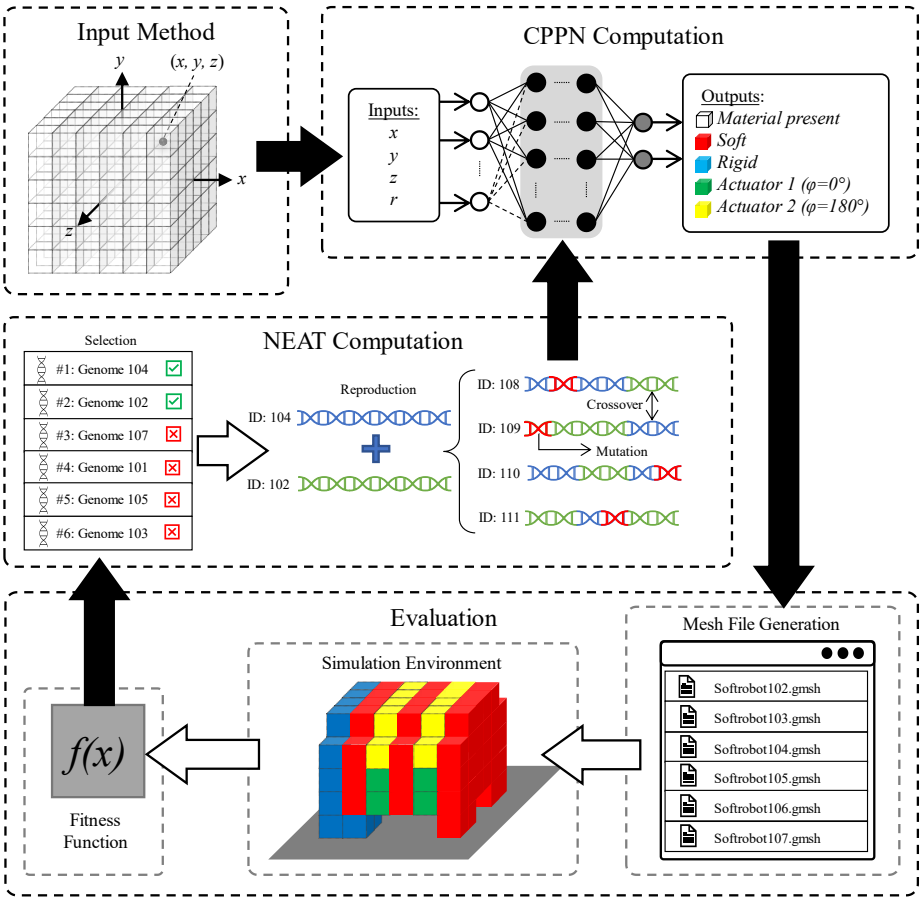


Figure 27: Methods Diagram

5.2 Input Methodology

To avoid excessive computation, the number of elements that the soft robot can be made from has to be limited. This can make the order in which the positional information is fed into the CPPNs important and contributes to the shape of the soft robots generated. In previous research, positional information was fed into CPPNs by iterating through a three-dimensional grid of elements. This process is shown using the pseudocode in Algorithm 5.1.

Algorithm 5.1: Grid Mesh Generation Algorithm

```
function CREATE ELEMENT DICTIONARY (width, length, height):
    set element number = 0                          ► Used as the count and element label
    for (x, y, z) between (0, 0, 0) and (width, length, height) do:
        set centre point = (x, y, z)                ► Used as the input to the CPPN
        set list of vertices ← store the vertices of the element at x, y, z   ► Used to create the mesh file
        set element dictionary [element number] = [centre point, list of vertices]
        increment element number by 1
    end for
    return element dictionary
end function

set element dictionary ← CREATE ELEMENT DICTIONARY (10, 10, 10)
set THRESHOLD = 0.2
initialise list of soft elements ← stores elements that will be made of soft, elastic material
initialise list of rigid elements ← stores elements that will be made of rigid material
initialise list of actuator 1 elements ← stores elements that will make up the actuator with 0° phase angle
initialise list of actuator 2 elements ← stores elements that will make up the actuator with 180° phase angle
for element in element dictionary do:
    query CPPN with the centre point of the element, which returns:
        material present ← value between -1 and 1 stating if the element should be used
        soft material ← value between -1 and 1 stating if the material should be soft, elastic material
        rigid material ← value between -1 and 1 stating if the material should be rigid material
        actuator 1 ← value between -1 and 1 stating if the element be part of the actuator with 0° phase angle
        actuator 2 ← value between -1 and 1 stating if the element be part of the actuator with 180° phase angle
    if material present greater than THRESHOLD do:
        if soft material is the highest value of the outputs do:
            add list of vertices of the element to list of soft elements
        else if rigid material is the highest value of the outputs do:
            add list of vertices of the element to list of rigid elements
        else if actuator 1 is the highest value of the outputs do:
            add list of vertices of the element to list of actuator 1 elements
        else do:
            add list of vertices of the element to list of actuator 2 elements
        end if
    end if
end for
```

In this research, another input method was proposed, where the algorithm began from a central element placed at the origin. The position of this element was inputted into the CPPN and if the first output is above the threshold, then the positions of all adjacent elements are also inputted into the CPPN. These new elements were sorted by the value of their first output from highest to lowest and added to the queue, dictating the order in which they are added to the soft robot provided they exceed the threshold. This process was then similarly repeated for the adjacent elements to the elements in the queue. Every time this process is repeated elements are placed into the queue based on the value of the first output of the CPPN. In this way, the first output value not only dictates whether material is present at a specific position but also controls the direction that the mesh grows. This can be likened to morphogen concentrations seen in developmental biology. This process is described in Algorithm 5.2.

Algorithm 5.2: Growing Mesh Algorithm

```
set MAX NUMBER OF ELEMENTS = 1000
set STARTING ELEMENT = (0, 0, 0)
set THRESHOLD = 0.2
initialise queue ← stores list of elements to be inputted into the CPPN
add STARTING ELEMENT to queue
initialise list of soft elements ← stores elements that will be made of soft, elastic material
initialise list of rigid elements ← stores elements that will be made of rigid material
initialise list of actuator 1 elements ← stores elements that will make up the actuator with 0° phase angle
initialise list of actuator 2 elements ← stores elements that will make up the actuator with 180° phase angle
set number of added elements = 0          ► Count to determine how many elements have been added
set element = 0                           ► Index of the current element in the queue
while number of added elements is less than MAX NUMBER OF ELEMENTS do:   ► Stops after 1000 elements
    query CPPN with the centre point of the next element in queue, which returns:
        material present ← value between -1 and 1 stating if the element should be used
        soft material ← value between -1 and 1 stating if the material should be soft, elastic material
        rigid material ← value between -1 and 1 stating if the material should be rigid material
        actuator 1 ← value between -1 and 1 stating if the element be part of the actuator with 0° phase angle
        actuator 2 ← value between -1 and 1 stating if the element be part of the actuator with 180° phase angle.
    if material present is greater than THRESHOLD do:
        set list of vertices ← compute and store list of vertices for the current element
        compute adjacent elements
        insert adjacent elements into queue ordered by material present from highest to lowest
        if soft material is the highest value of the outputs do:
            add list of vertices of the element to list of soft elements
        else if rigid material is the highest value of the outputs do:
            add list of vertices of the element to list of rigid elements
        else if actuator 1 is the highest value of the outputs do:
            add list of vertices of the element to list of actuator 1 elements
        else do:
            add list of vertices of the element to list of actuator 2 elements
        end if
    end if
    increment element by 1                ► Change index to move to the next element in the queue
end while
```

5.3 CPPN-NEAT Implementation

The python library, neat-python, was used to perform the evolutionary computations, such as selection, reproduction and speciation. In addition, neat-python creates the CPPNs and computes the outputs using the CPPNs. Two aspects of the evolution were required by the user: an evaluation function and a configuration file. The evaluation function consisted of the generation of the mesh files, the simulation of the soft robots, and the fitness function. These are discussed in the subsequent sections. The

configuration file contains all the settings of the evolutionary process, the settings used for this research is shown in the configuration file in the digital appendix which are the same as those used in previous research with some minor modifications.

5.4 Mesh File Generation

SOFA is compatible with many different mesh file types including Wavefront OBJ file formats (.obj), Visualisation Toolkit file formats (.vtk), stereolithography CAD files (.stl), Object File Formats (.off), and GMSH file formats (.gmsh). The GMSH file format was used to define the soft robots meshes, due to the simplicity of the format. As can be seen, the simple layout allowed for each element of the soft robot to be written to the file automatically using a simple python function. Each material of the soft robot required its own mesh file. Each mesh file was saved using the unique genome ID and the material type, making it simple to import the file into the SOFA scene. To correctly generate the mesh files, it was necessary to store and track the vertices, vertex numbers and element numbers during the CPPN querying process. In addition, any shared indices between different materials of the soft robot had to be stored and correspond to their respective vertex number within each mesh file. This was necessary to define which indices of the one part attached to the indices of the other part in SOFA's attachment constraint. This process is shown in Algorithm 5.4.

Algorithm 5.4: Mesh File Generation

INPUTS: *list of soft elements*, *list of rigid elements*, *list of actuator 1 elements*, and *list of actuator 2 elements*
repeat for each input **do:**
    initialise *list of soft part's vertices* ← stores the coordinates of all the vertices of the soft part
    for *element* in *list of soft elements* **do:**
        for *vertex* in *element* **do:**
            add *vertex* to *list of soft part's vertices*
        end for
    end for
    remove duplicates from *list of soft part's vertices*
    sort *list of soft part's vertices*
    for *element* in *list of soft elements* **do:**
        for *vertex* in *element* **do:**
            get *index* for *vertex* in *list of soft part's vertices*
            replace *element* in *list of soft elements* with the *index* number
        end for
    end for
    initialise *soft part mesh file path* using the soft robot's genome ID ← location that the mesh file will be stored.
    create *soft part mesh file* in the *soft part mesh file path*
    write '$NOD' to *soft part mesh file*
    write the length of *list of soft part's vertices* on the next line in *soft part mesh file*
    for *vertex* in *list of soft part's vertices* **do:**
        write *index* of *vertex*, x coordinate of *vertex*, y coordinate of *vertex*, and z coordinate of *vertex* on the next line in *soft part mesh file*
    end for
    write '$ENDNOD' on the next line in *soft part mesh file*
    write '$ELM' on the next line in *soft part mesh file*
    write the length of *list of soft elements* on the next line in *soft part mesh file*
    for *element* in *list of soft elements* **do:**
        write *index* of *element*, '5 1 1 8', 1st vertex in *element*, 2nd vertex in *element*, 3rd vertex in *element*, 4th vertex in *element*, 5th vertex in *element*, 6th vertex in *element*, 7th vertex in *element*, and 8th vertex in *element* on the next line in *soft part mesh file*
    end for
    write '$ENDELM' on the next line in *soft part mesh file*
    close *soft part mesh file*
end repeat
repeat for each combination of materials **do:** ► The soft part and rigid part is used as a sample
    initialise *soft to rigid connection* ← stores the indices of the vertices from the *list of soft part's vertices* that are also present in the *list of rigid part's vertices*
    initialise *rigid to soft connection* ← stores the indices of the vertices from the *list of rigid part's vertices* that are also present in the *list of soft part's vertices*
    for *soft vertex* in *list of soft part's vertices* **do:**
        for *rigid vertex* in *list of rigid part's vertices* **do:**
            if the coordinate of *soft vertex* equals the coordinate of *rigid vertex*, **then:**
                add the index of *soft vertex* to *soft to rigid connection*
                add the index of *rigid vertex* to *rigid to soft connection*
            end if
        end for
    end for
end repeat

5.5 Set-up of Simulation Environments

All experiments are done using Simulation Open Architecture Framework (SOFA). However, the performance of evolving soft robots in SOFA was compared to that in VoxCAD used in previous research. Although open access to the results of previous work was provided, their runs used unrealistically high actuation periods making it unusable for comparison, thus evolutionary runs were also done in the VoxCAD environment in this research with the same simulation and evolution settings to allow for comparisons to be made.

5.5.1 Set-up of VoxCad

Evolutions were performed in VoxCad by using the evosoro python library, which was built on top of VoxCad and uses the underlying Voxelyze physics engine. Since all the evolution and simulation was already set up in this library, only the evolution settings and environment parameters needed to be specified, which are specified in Sections 4.3 and 4.5.3, respectively. Importantly, Figure 31 shows how the parts of a soft robot are defined in VoxCad, which is used to convert soft robots generated in SOFA to VoxCad and vice versa.

5.5.2 Set-up of SOFA

SOFA has been used to simulate soft robots before, thus many plugins to aid in the simulation of soft robots are available. In addition, the development of a python 3 plugin was completed allowing for the easy integration of the neat-python library, other python libraries and SOFA. As described in Section 2.4.4, the simulation uses a modular simulation structure, which allows for different aspects of the simulation to be simulated with different models with varying degrees of resolution and efficiency. The structure of a simulation is represented using a Directed Acyclic Graph (DAG) or a tree, where different objects in the simulation are represented as different nodes, and different models of an object are contained within this node as sub-nodes. In this research, two environments are used to validate results, and to gain a better understanding of the difference between all methods compared. The two environments that are used are a terrestrial environment and an underwater environment.

Terrestrial Environment

The terrestrial environment consists of a flat surface with a soft robot placed at the centre. This environment requires the modelling of not only the physics models of the soft robot but also a collision model to simulate the interaction between the soft robot and the flat surface. It is also important to model the friction between the two objects, which can be included in the collision model in SOFA. The DAG for the simulation of soft robots in a terrestrial environment is shown in Figure 28.

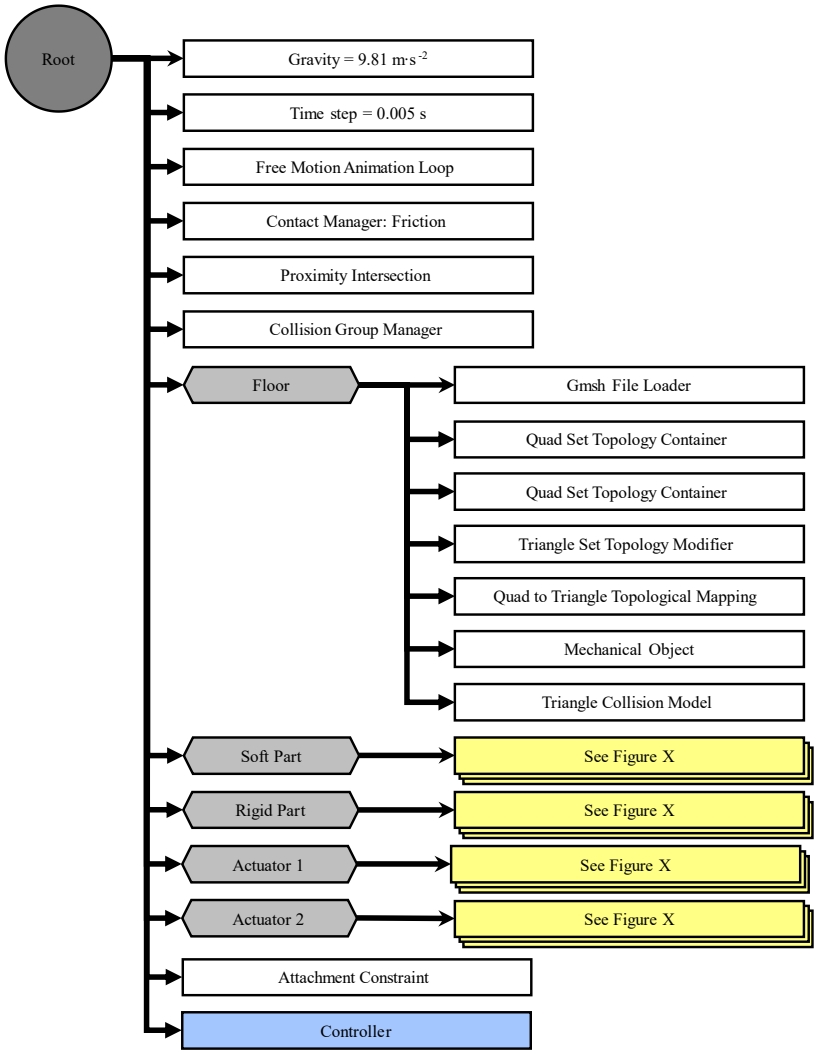


Figure 28: The scene graph of simulation environment used to evaluate soft robots during the evolutionary process.

The following assumptions were made:

- The friction coefficient was constant between all the materials.
- All materials were assumed to deform according to Hooke's law.

Controller

As can be seen in the set-up of the environment, a controller was added to the scene graph. The controller was used to control the pressure inside the actuators during the simulation. Since the pressure-driven actuators are heavily dependent on the structure of the soft robot, for the same pressure the amount of deformation varies significantly. To overcome this, in the first 5 seconds of each simulation, the controller slowly increased the pressure each time step until the actuator increased in volume by 20 percent, which is the volumetric actuation used in the previous work. In addition, during these 5 seconds, the gravity is slowly increased in the terrestrial environment to gently place the soft robot on

the flat surface without any falling or rolling. After this 5 second initialisation period, the actuators are actuated sinusoidally out of phase with each other for 15 actuation cycles. In the underwater environment, the drag is only enabled after this initialisation period, otherwise, the evolution would evolve soft robots that only utilised the initial boost from inflating its actuators during this initialisation period as opposed to using the actuator during their actuation cycles to propel forward.

5.5.3 Material Properties

The material properties used for the evolutions are shown in the digital appendix. It is important to note that these properties were based on some previous work [1]. In addition, the term rigid material refers to the stiffer material but is not necessarily rigid compared to other materials such as metals or ceramics. The low elastic modulus was used for the rigid material such that it is still capable of being bent and stretched by the actuators. Particularly, the actuator stiffness skin stiffness was determined first through a calculation based on an assumed thickness, and then iteratively adjusted based on preliminary results. This was done to ensure that the actuator was not too stiff such that no passive support material would be used, but not too soft that the actuator would collapse on itself resulting in unrealistic physics.

5.5.4 Domain Randomisation

Domain randomisation was implemented by randomising the material and environment properties inside the SOFA scene by a certain percentage of its original value. In this work, a small percentage of one percent was used, since larger values would result in no convergence being reached. The following properties were randomised:

- Gravitational acceleration.
- Friction constant between the robot and the surface.
- Young's moduli of the materials.
- Poisson's ratio
- Densities of the materials.

6 RESULTS AND DISCUSSION

In this section the results of the methods described in Chapter 5 are presented and discussed. In each section, observations on both the physical structure and behaviour of the soft robots are presented and discussed. In relation to the structure of the soft robot, the material composition and the shape of the soft robot are the key aspects that were focused on. Observations about the behaviour of the soft robot include categorising the type of motion used by the soft robot to propel itself forward such as jumping or galloping, as well as, how the soft robot achieves this motion. For example, for a soft robot to propel itself forward on land, it needs to shift its weight and momentum in a particular direction. This momentum shift can be accomplished in a variety of ways. Following the observations made on the evolved morphologies of the soft robots, the results obtained from the analyses and comparisons described in Chapter 5 are presented.

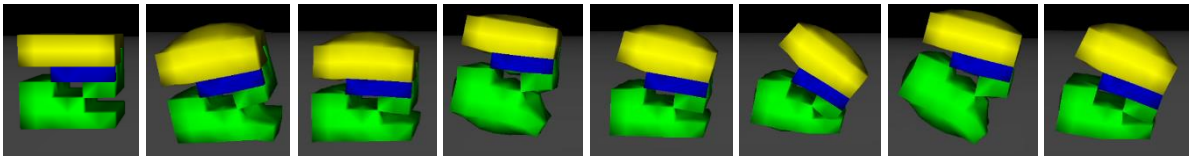
6.1 Validation of Modifications

Before core analyses were performed in this research, it was first important to analyse the minor modifications made to the set-up used. As stated in Section 5.1, modifications were made to the settings of NEAT and to the fitness function used for the evolution of a soft robot on a flat terrain.

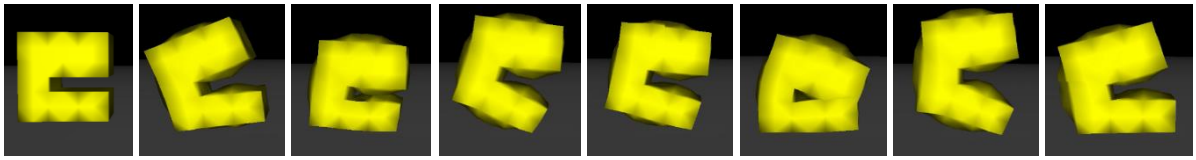
6.1.1 NEAT Settings and Fitness Function used in this Work

To compare the setting and fitness function used in this work before performing computationally expensive evolutionary runs, smaller evolutionary runs are made where the size of the mesh is limited to 6 by 6 by 6 elements cube. Figure 29 shows the soft robots with the highest fitness at the end of the evolutionary run for 5 independent runs using the settings and fitness function used throughout this research. All the best performing soft robots propel themselves forward by jumping except for one soft robot (Figure 29e) that utilises a motion that can be likened to galloping. Three of the jumping robots (see Fig. 29a, 29c, and 29d) achieved a much higher distance than the galloping soft robot indicating that this type of motion is probably more suitable for this environment at this scale with a maximum length of 90 mm. The soft robots evolved in seed 2 and 4 are both solely made-up of actuating material and both use the same method of propelling themselves forward. These soft robots both contain groove at the front of the soft robot causing the robot to lean forward when in contact on the ground. As the actuator inflates, the part below the groove kicks the soft robot up and forward. The location of the groove and slight changes to the shape of the soft robot drastically improves the performance of the soft robot evolved in seed 4 compared to the soft robot in seed 2. The evolved soft robots in seed 1 and 3 also use a structure to shift their momentum, but also contain rigid parts in the upper section. In seed 1, the rigid part is only situated in the front half of the soft robot, and since the rigid material is denser than the actuating part, the soft robot leans forward when in contact with the ground resulting in a forward jumping motion every time the actuator inflates. In seed 3, the combination of actuating

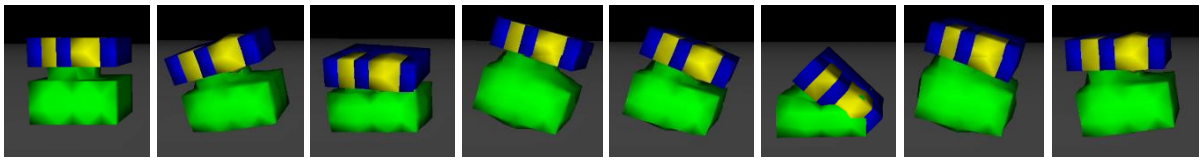
material and rigid material results in a rocking back-and-forth motion of the top half, during the jumping motion. As can be seen in seeds 1-4, the evolved soft robots take advantage of the lack of self-collision in the simulation set-up to achieve their jumping motion, which may be a potential drawback to the current simulation set-up. These results show that the evolutionary set-up used in this work can successively evolve many unique soft robotic designs capable over moving across a flat terrain at this scale and resolution. Importantly, this is the first known case where fluidic actuators have been used as the actuators in evolving soft robots.



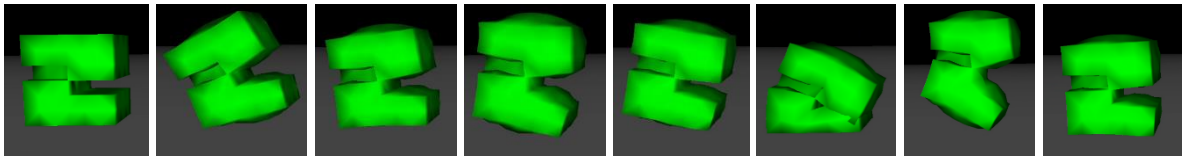
(a) Seed 1



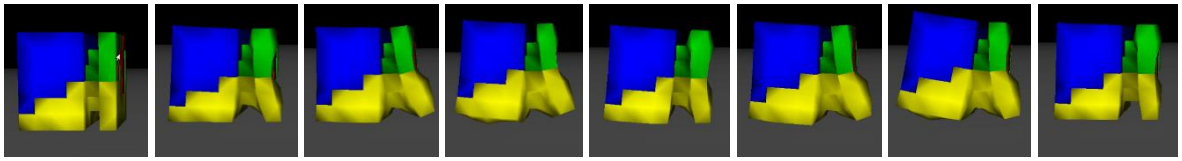
(b) Seed 2



(c) Seed 3



(d) Seed 4

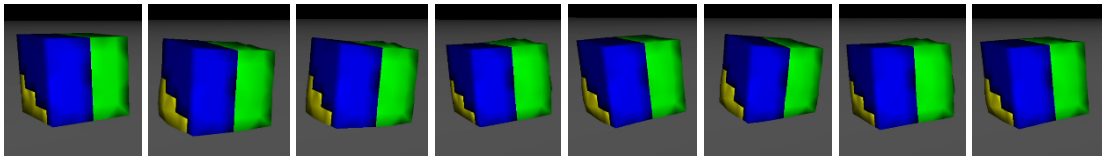


(e) Seed 5

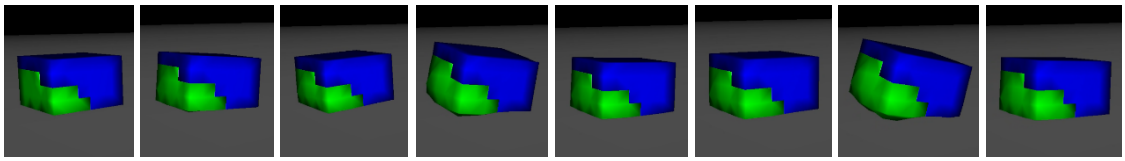
Figure 29: The motion of the top performing soft robots evolved from 5 independent runs, where each frame is captured at 0.3 second intervals. Videos for each can be found in the Digital Appendix.

6.1.2 NEAT Settings used in Previous Literature

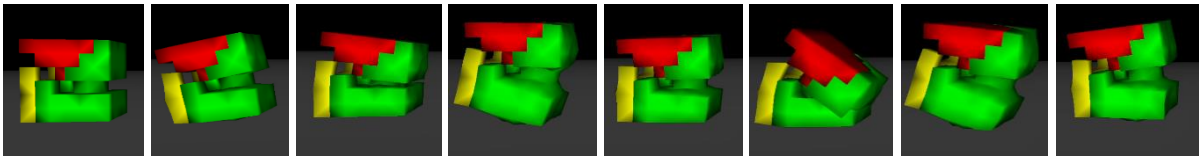
The primary reason modifications were made to the evolution settings was to increase the exploration of the solution space to account for the reduction in the number of generations during the evolution. This adjustment was necessary due to the increase in the computational power and time required by using a more realistic simulation environment. These modifications include increasing the probability of nodes and connections being added, removed, and changed. Figure 30 shows the best performing soft robots evolved in five independent runs. Using these settings from previous literature there is no considerable difference in the performance of the soft robots. In terms of behaviour, the soft robots evolved in seeds 3 and 4 both behave in similar way to those evolved using the modified settings used in this work, where soft robots jump in a particular direction due to a groove situated at the front of the soft robot. The soft robot evolved in seed 3 enhances this motion with a more complex groove structure and the use of an opposing actuator at the back of its body, which inflates as the robot lands causing the soft robot to shift its momentum forward. Seeds 1 and 2 edge use a different method of moving forward, where an actuator at the back bottom corner of its body incrementally pushes the soft robot forward as it inflates. The soft robot evolved in seed 1 optimises this structure and motion with an additional opposing actuator such that the robot rocks back on forth as it pushes itself forward. These soft robots perform substantially worse than most of the jumping robots, further indicating that jumping is the optimal behaviour for this environment for robots of this size. The soft robot evolved in seed 5 is the only evolved soft robot which does not develop a consistent motion and falls over despite the use of the modified fitness function designed to penalise falling and inconsistent motion. This result can be attributed to two reasons, soft robots that move for a long enough time before falling over is still able to obtain a high fitness particularly if the falling motion is slow, and the simulation time is not long enough allowing soft robots with consistent motions to surpass falling robots. This allowed a soft robot with this type of behaviour to obtain a good fitness during the run time.



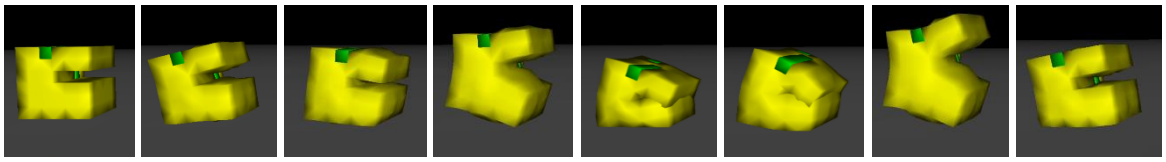
(a) Seed 1



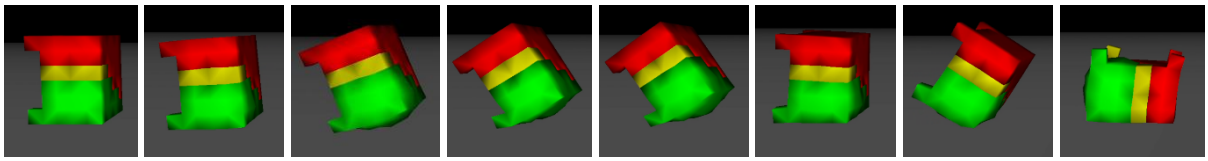
(b) Seed 2



(c) Seed 3



(d) Seed 4

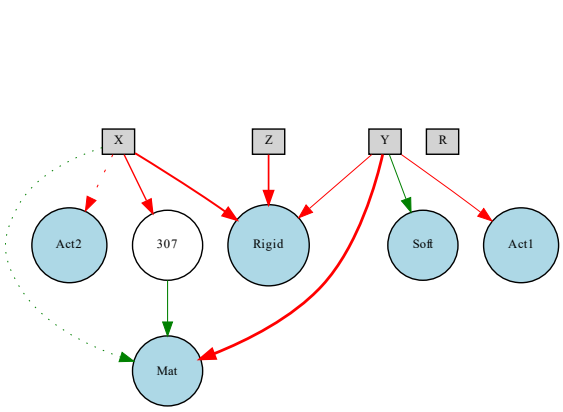


(e) Seed 5

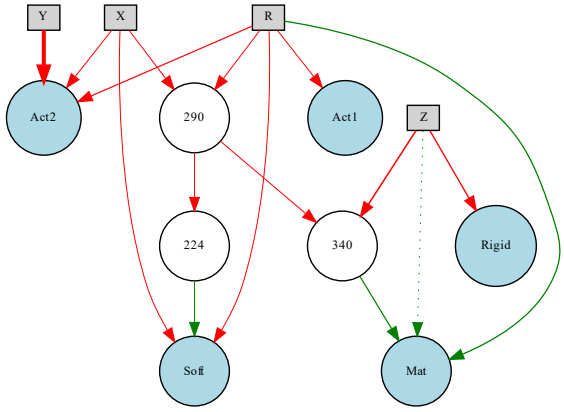
Figure 30: The motion of the top performing soft robots evolved from 5 independent runs using NEAT settings from previous literature, where each frame is captured at 0.3 second intervals. Videos for each can be found in the Digital Appendix.

The most noticeable difference between the soft robots evolved using the modified settings in this work and the soft robots evolved using previous literature's settings, is that the former have more complex body shapes. This may be indicative of the evolved CPPNs having more nodes and connections. This observation is confirmed when looking at the CPPNs of the best performing soft robots. Figure 31 shows both the evolved CPPNs with the minimum and maximum number of connections and nodes. As can be seen, using the settings from previous works the evolution optimised the weights of the connections effectively, but failed to find optimal solutions beyond the initial CPPN structure of 5 nodes and 10 connections. Interestingly, using the modified settings, four out the five of the evolved CPPNs using the modified settings have more hidden nodes connecting to the material output node than other nodes. This explains the observed difference in the complexity of the shapes of the evolved soft robots using the different settings. The convergence on having these hidden nodes suggests that how beneficial

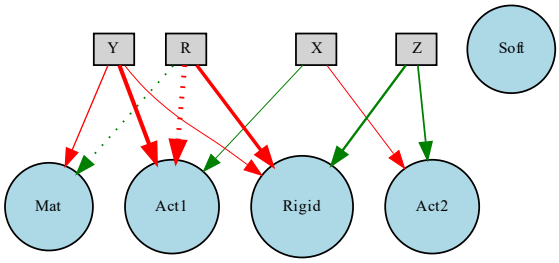
more complex shapes can be on the performance of the soft robot without necessarily having complex internal material structures and connections. The material output node corresponds to whether material is present at a position or not thus dictating the shape of the soft robot.



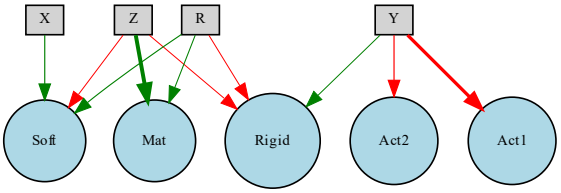
(a) Minimum complexity using Modified Settings



(b) Maximum complexity using Modified Settings



(c) Minimum complexity using Old Settings



(d) Maximum complexity using Old Settings

Figure 31: The CPPNs with the minimum and maximum complexities evolved using the modified NEAT settings and the NEAT settings used in previous work

It is clear that the modification to the settings does manage to evolve soft robots with more complex CPPN structures than the initial defined structure during 200 generations, whereas the old settings failed do so. This is important such that the evolution is less dependent on the initial conditions of the evolutionary set-up and effectively explores the solution space. To further validate this result, Figure 32 shows that these modifications successively increase the amount of exploration of the evolution. Evolution performed with the modified settings results in a median of 58 different CPPN structures in 200 generations, whereas evolution performed with the settings used in past literature only explored a median of 17 CPPN structures during the same number of generations.

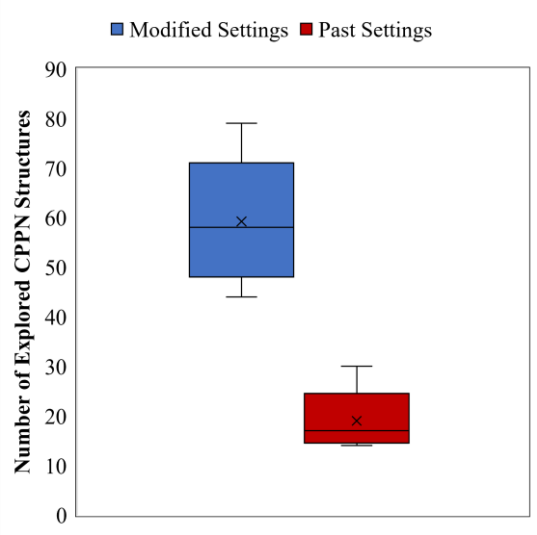
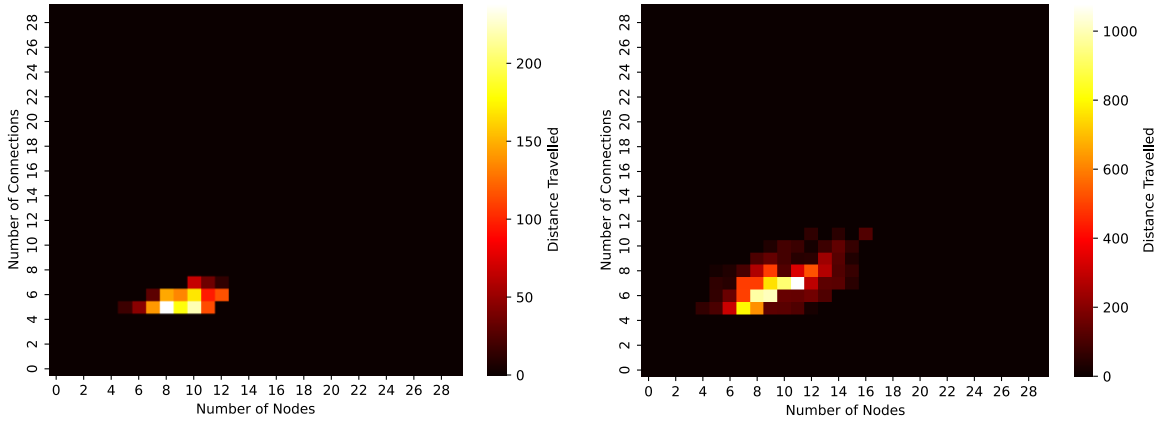


Figure 32: Comparison of the amount of explored CPPN structures as a result of NEAT settings used in previous research versus that used in this research.

Furthermore, Figure 33 shows the median exploration of solution space in terms of CPPN structures as heatmaps. Interestingly, despite the significant increase in the exploration, the best performing structures using both settings are not substantially different in size and complexity. As stated, mostly only one or two hidden nodes were only required to effectively improve the shapes of the soft robots, indicating that the increase in exploration might not have necessarily improved the performance of the evolution in this task. Despite this observation it has been proven with substantial evidence that increasing the exploration of the solution space leads to improved performance of evolutionary algorithms particularly in overcoming local maxima to find global maxima, which may be paramount in more challenging soft robotic tasks and environments.



(a) (b)

Figure 33: Heatmaps showing the exploration of the solution space in terms of CPPN structures as well as the performance of different structures. (a) The exploration of the evolution using settings from previous literature. (b) The exploration of the evolution using the settings in this research.

Figure 34 confirms that the increase in exploration did not improve the performance of the evolution, as no significant difference is found in the performance between the two evolutions settings used. An additional reason as to why the increase in exploration of the solution space may not have improved the performance is that although it aids in finding completely different solutions thus overcoming local maxima, it does not perform as well at making smaller modfications to improve existing solutions. With results indicating that fairly simple CPPN structures and soft robots having a high performance, this can be a problem for this environment and task.

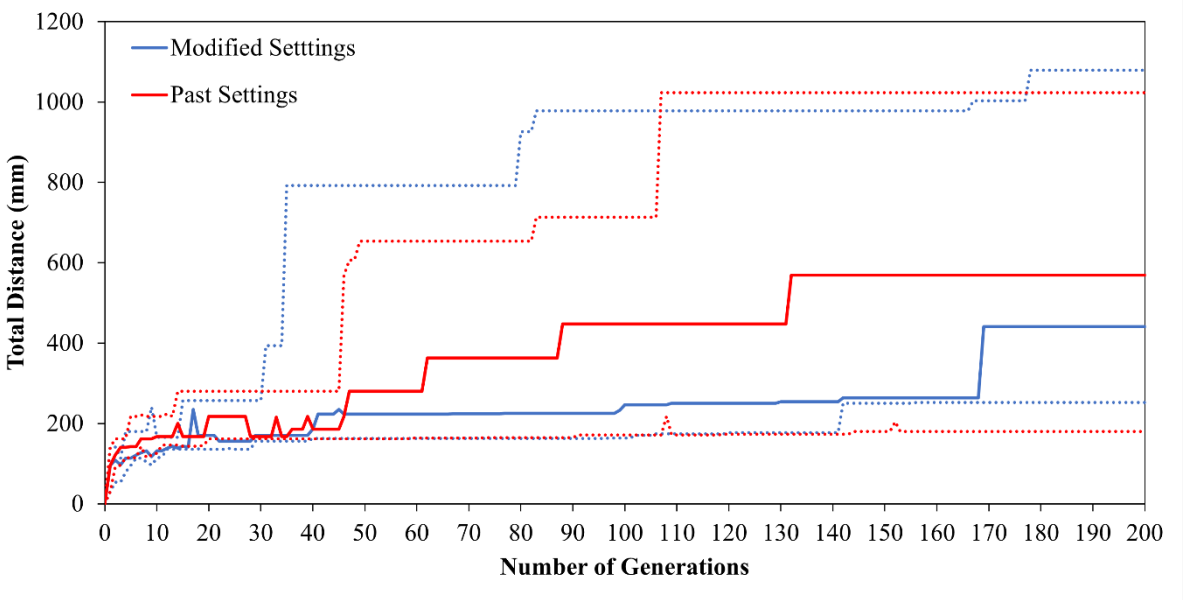
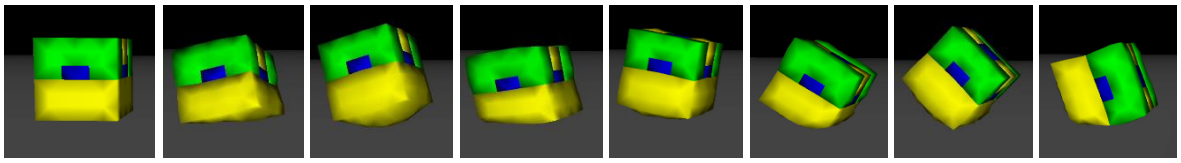


Figure 34: The total distance achieved by the highest fitness soft robots for 5 independent runs for both NEAT Settings used. The solid lines represent the median distances, and the dashed lines represent the ±95% bootstrapped confidence intervals.

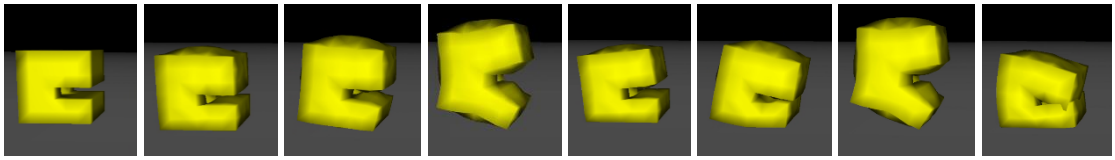
6.1.3 Fitness Function used in Previous Literature

Figure 35 shows the behaviours of the evolved soft robots from five independent runs using the fitness function from previous literature – the total distance travelled by the soft robot. Three of the five evolutionary runs successively evolved a soft robot that can consistently propel itself forward using actuators. All three of these soft robots use a similar design as seen in the previous results, where the soft robot leans forward due to a groove at the front of the body and jumps using a large fluid elastomer actuator. There are only minor compositional differences between them. The other two evolutionary runs failed to produce a soft robot that can consistently propel itself forward. The evolved soft robot in seed 1 rocks back and forth before it falls over, and the evolved soft robot in seed 3 similarly rocks back and forth during its actuation cycle but failed to move in any direction consistently. Both these soft robots would have obtained substantially worse fitness values using the modified fitness function, thus driving the evolution away from these solutions towards more consistent motions even if their total distance covered is less. This is further confirmed by the absence of alternative behaviours to jumping

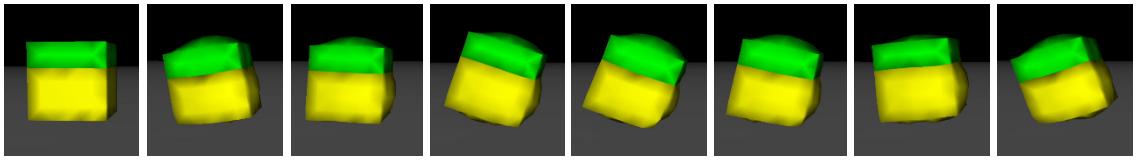
such as the galloping and pushing motions seen in Sections 6.1.1 and 6.1.2. Although these behaviours are less effective than jumping, they are extremely consistent, which would make them significantly easier to control and make them more reliable in different environments.



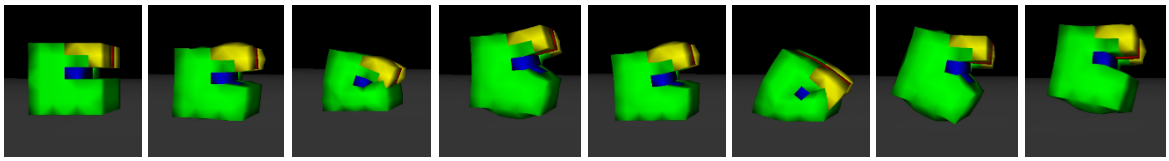
(a) Seed 1



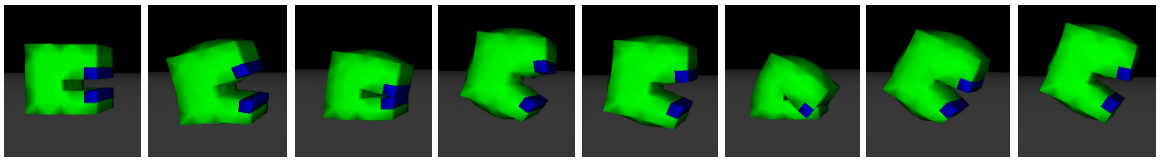
(b) Seed 2



(c) Seed 3



(d) Seed 4

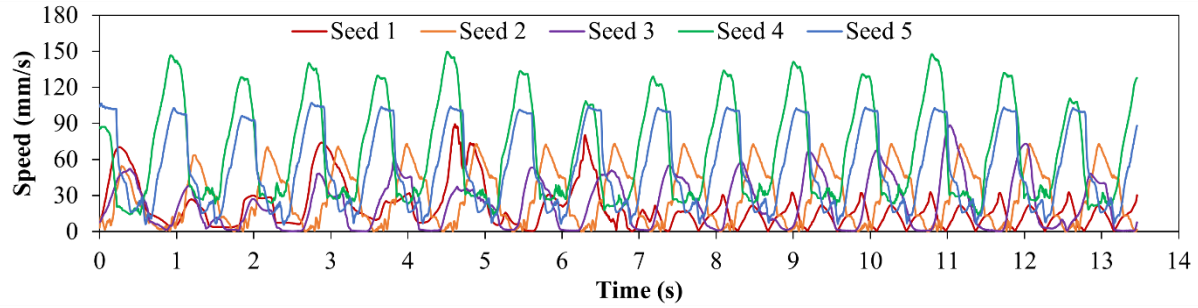


(e) Seed 5

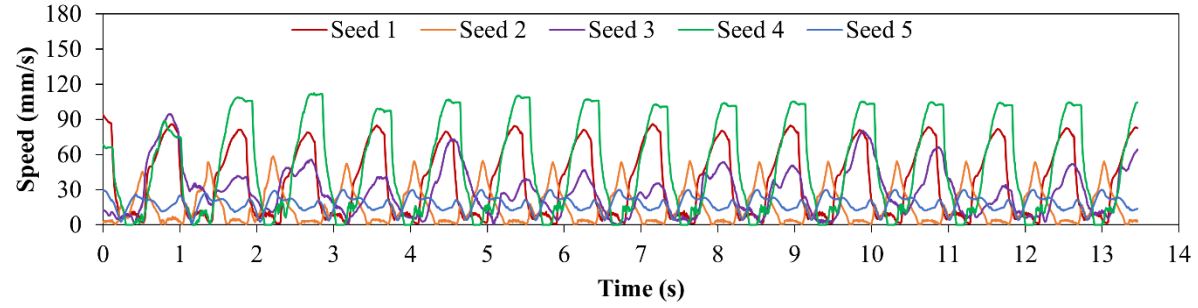
Figure 35: The motion of the top performing soft robots evolved from 5 independent runs using the fitness function used in previous literature, where each frame is captured at 0.3 second intervals. Videos for each can be found in the Digital Appendix.

To show the consistency of the movement of the soft robots, both the speed versus time and the paths of the evolved soft robots using the two fitness functions were compared. Figure 36 shows the speed of the soft robot versus time of 5 independent runs for each fitness function used. All the soft robots evolved using the modified fitness function (Figure 36b) move with a consistent periodic motion. Interestingly, 4 out of the 5 soft robots evolved using total distance as the fitness function (Figure 36a) also moved with consistent periodic motion, but one of these four soft robots simply rock back and forth as stated earlier. One difference between this set-up and that in previous literature is that the soft robots

are simulated for 15 actuation cycles after a 5 second initialisation period. The five second initialisation period allows for robots to fall or roll without it being included into its fitness. Additionally, fifteen actuation cycles make it more likely that consistent movements in a certain direction should be able to surpass the distances of fallen soft robots given the extra five actuation cycles. This is likely why despite using the total distance as the fitness function, the evolution was still able to produce successful soft robots in three out of the five independent runs. However, as can be seen in Figure 36 that in seed 1, the evolve soft robot moves with sudden movements up until around seven seconds, where thereafter it has minimal movement. Importantly this indicated that if the soft robot operated for longer periods of time, the distance it would achieve would not increase compared to those with consistent motion.



(a)



(b)

Figure 36: The speed of the soft robots during the evolution for 5 independent runs. (a) Using total distance as the fitness and (b) using the average of the lowest two distances travelled during 5 actuation cycles as the fitness.

Figure 37 further confirms the observations and results stated thus far, showing the paths taken by the top performing soft robots using the two different fitness functions. The use of the modified fitness function results in soft robots that not only cover a larger distance using periodic motion, but also evolves soft robots that move along straighter paths compared to some of those obtained using the total distance as the fitness function (Seeds 1, 2 and 3 in Figure 37a). In Figure 37a, it is seen that the soft robots in seeds 1 and 3 perform worse in terms of distance than the soft robots with period consistent motion, yet these behaviours were not discovered in these seeds. This is caused by the evolutions inability to get out of this local maximum. Conversely, this was not the case with the modified fitness

function as these behaviours would have obtained substantially worse fitness values using the modified fitness function.

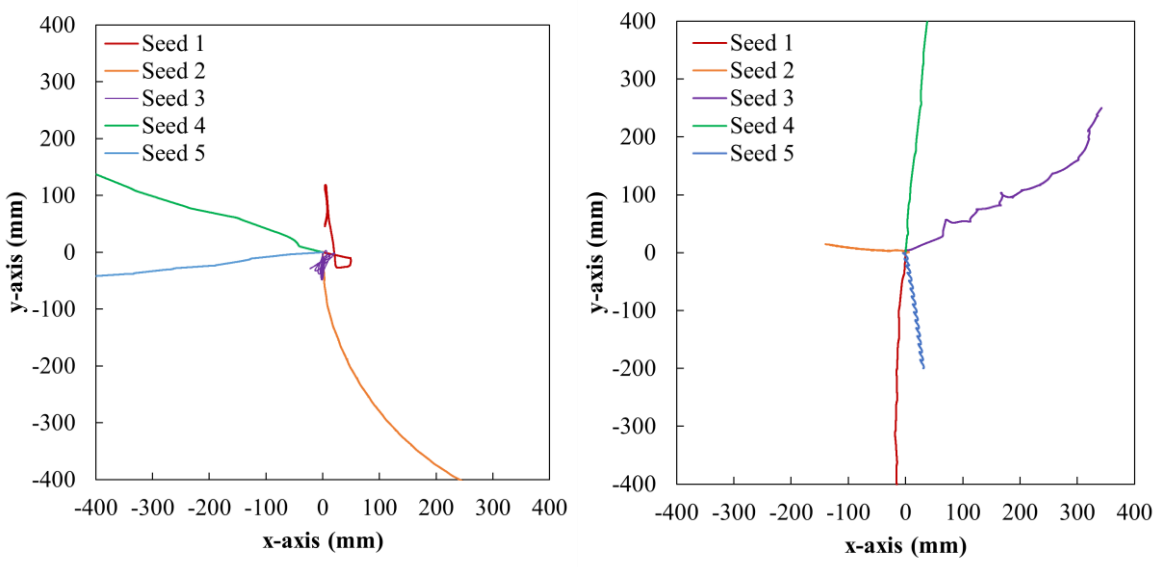


Figure 37: Top-view showing the paths taken by the soft robots during 15 actuation cycles

In Figure 38, it is seen that the evolutions perform slightly worse when using the modified fitness function, but the difference is non-significant. However, Figure 38 indicated that most soft robots with consistent periodic motion does perform better than those with sudden movements, so it would be expected that the evolutions performed with the modified fitness function would perform better. One possible drawback of the modified fitness function limiting its performance is that it may penalise soft robots with imperfect motion that may be steppingstones to soft robots with even better performances. This concept is introduced by Gaier et al. [32], where it is shown that CPPNs can be evolved to generate a target image using MAP-Elites. As described in Section 2.3.3, MAP-Elites searches for solutions in different niches making it very effective exploring the solution space. Since MAP-Elites encourages finding different solutions, it is surprising that this method is capable of converging onto solutions that closely resemble the target image. This is despite the inability of this target image to be obtained evolving CPPNs where their fitness is determined by directly comparing its generated image with the target image. The justification for this result is that it can be useful to first explore different solutions as these might make it easier to then obtain the desired solutions, thus acting as steppingstones to the desired image. Similarly, a soft robot which may move in a circular motion initially, could lead to better soft robots that move in straight lines with only small modifications to the CPPN.

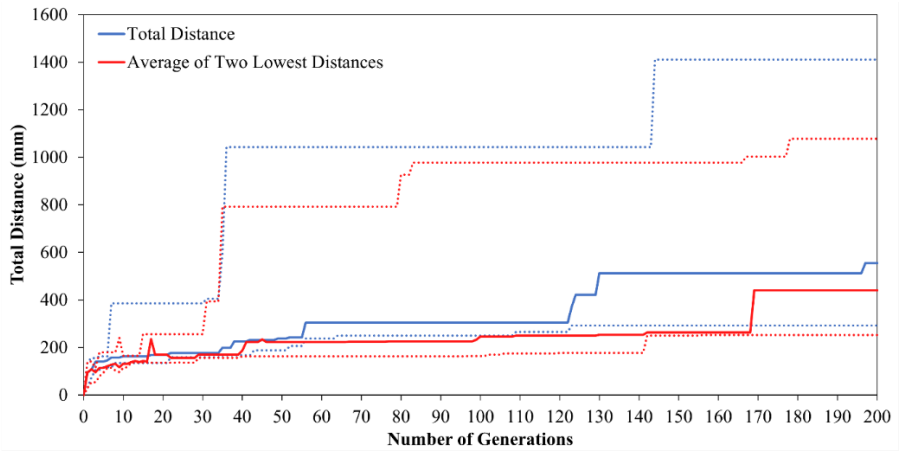
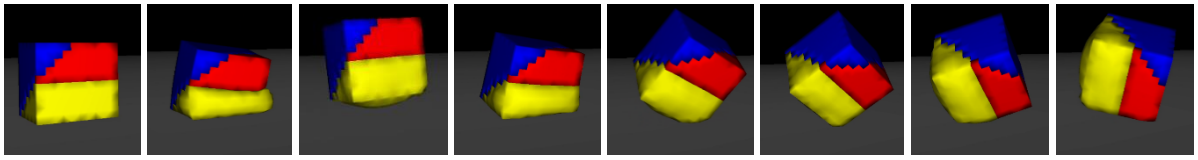


Figure 38: The median distance travelled by the top-performing soft robots using the two fitness functions. The solid line represents the median value, and the dotted lines represent the 95% bootstrapped confidence intervals.

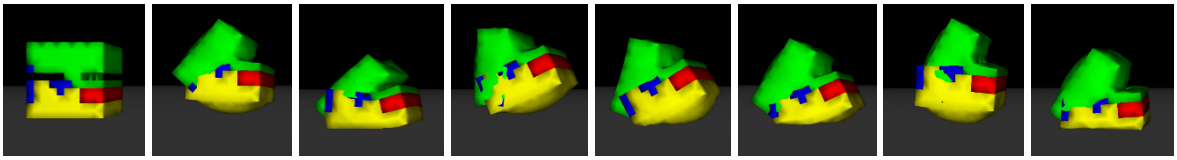
6.2 Effect of Domain Randomisation on Performance and Diversity

Before analyses are made on the effect of domain randomisation on the evolution, it is of interest to visually compare the morphologies of the evolved robots with that of results presented in Section 6.1 and between the fitness search, novelty search and fitness search with domain randomisation. Figure 39 shows the behaviour and structure of the best performing soft robots evolved using fitness search. Firstly, there is a noticeable difference in the realistic shape of the inflated actuator compared to results obtained in Section 6.1 due to the increase in resolution. Unlike the previous results in Section 6.1, jumping is the only behaviour observed in the top-performing robots, indicating that at this scale it may be even more dominant at this scale compared to other behaviours. The only difference between the best soft robots from the different runs is the method that it used to shift its momentum to jump in a particular direction. In seeds 1 and 4, the internal shape of its passive parts causes it roll slightly forward when it touches the ground. This occurs since the actuator beneath these parts are deflated and compressed under its own body weight. This can be clearly seen in the second and fourth frame of Figure 39a and in the third and fifth frame of Figure 39d. Unfortunately, in both cases, this method of locomotion results in the soft robots leaning too far forward during the jumping motion resulting it falling forward. Since the falling only happens towards the end of the simulation after consistent periodic motion, these soft robots still obtain high fitness value, and their final distance are even boosted by the falling motion in the direction it was going. Despite having very different internal structures, the soft robots evolved in seed 2 and 5 both exploit the lack self-collision to allow for a unique behaviour. Both soft robots consist of a top and bottom part connected by a narrow section. This results in the top part folding down through the bottom part such that it contacts the ground near the rear end of the soft robot's body. When these soft robots land on the ground, this upper part of the soft robot hits the ground creating a reaction moment causing the soft robot to shift its momentum forward before it takes its next

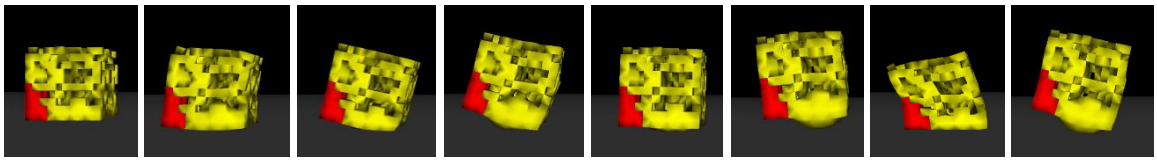
jump. Additionally, the evolved soft robot contains an interesting internal structure that resembles a PneuNet actuator. This design results in most of the actuation force being directed perpendicular to the actuators channels due to the higher surface area. In this case it results in a larger jumping force. Lastly, the evolved soft robot in seed through shows the amount of complexity that can be achieved with a relatively simple CPPN. Based on observation, the intricate sponge-like structure has two functions. Firstly, during the evolution the equivalent stiffness of this structure can be evolved to be different in every direction to improve the performance of the jumping motion. Secondly, since this structure is made up of a network of small channels, most of the actuating force and inflation is concentrated in parts with large volumes since larger surfaces deflect more under the same forces as smaller surfaces with the same elasticity. In this case, this occurs at the base of the soft robots, resulting in the jumping motion.



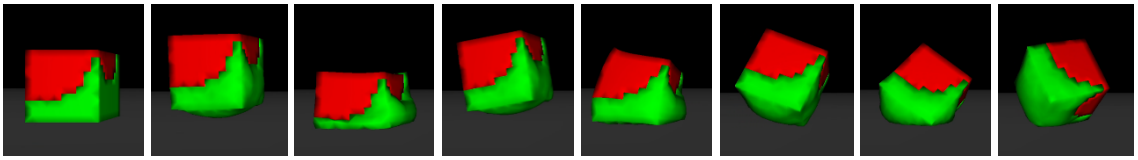
(a) Seed 1



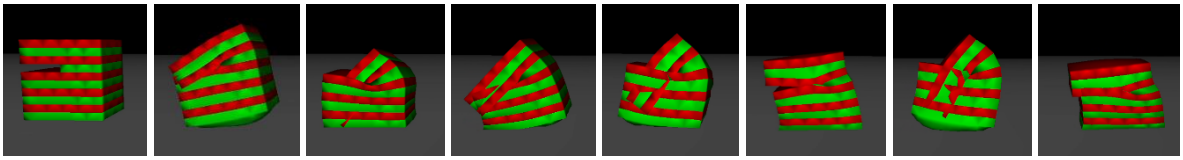
(b) Seed 2



(c) Seed 3



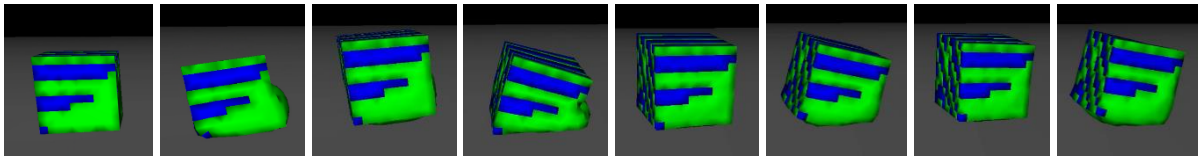
(d) Seed 4



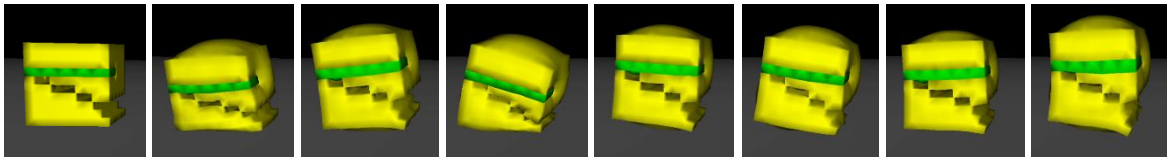
(e) Seed 5

Figure 39: The motion of the top performing soft robots evolved from 5 independent runs using fitness search. The first frame shows the initial shape, thereafter, each frame is captured at 0.5 second intervals towards the end of the simulation. Videos for each can be found in the Digital Appendix.

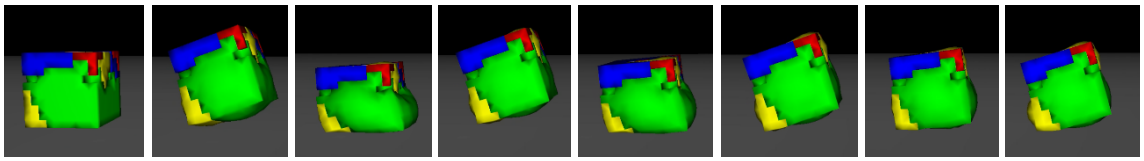
Figure 40 shows the best performing soft robots using novelty search from five independent runs. There is no observable difference in morphologies or behaviours between the two search methods. Once again, a channelled actuator structure resembling a PneuNet actuator (Seed 1 in Figure 40a); however, in this design it is used for a different purpose. The channelled structure provides flexibility along its width, which aids in its stability as it lands after every jump, whilst maintaining some rigidity in the direction of motion. This rigidity causes the soft robot to rock forward every time it contacts the flat surface similar to that seen in Figures 40a and 40d. Contrarily to those soft robots in Figure 40a and 40d, this soft robot does not fall over, which may be due to the reduced rigidity thanks to the channelled structure. The soft robots evolved in Seed 2 and 4, utilise similar structures and methods to jump as observed in previous results. Another structure used by previous results is observed in seed 3, where an opposing actuator is placed at bottom back corner of the soft robot such when the soft robot lands and the primary actuator that causes the jumping is deflated, the opposing actuator inflates resulting in the soft robot leaning forward onto the primary actuator. Lastly, the soft robot evolved in seed 5, uses a similar method, but instead of using an opposing actuator it uses a passive soft part shaped such that it causes the same leaning forward motion.



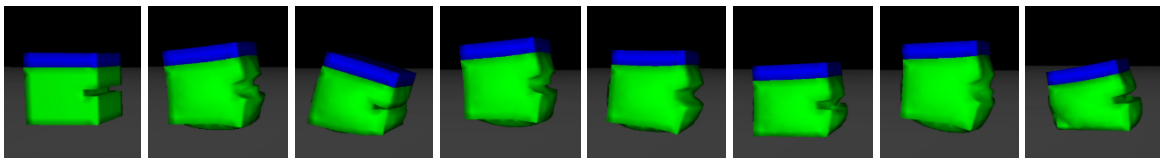
(a) Seed 1



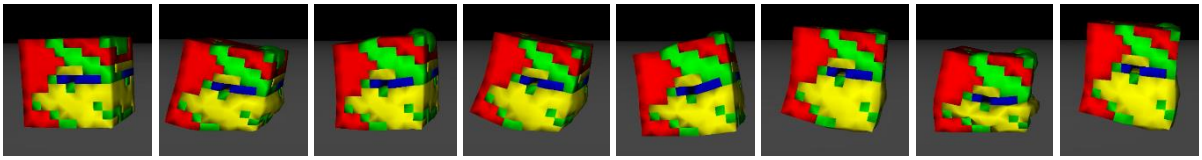
(b) Seed 2



(c) Seed 3



(d) Seed 4

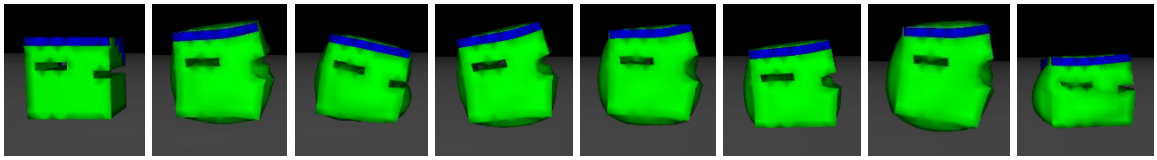


(e) Seed 5

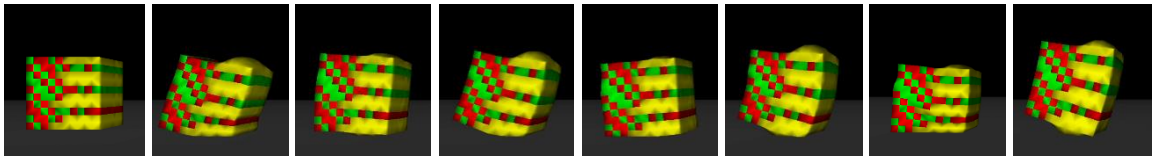
Figure 40: The motion of the top performing soft robots evolved from 5 independent runs using novelty search. The first frame shows the initial shape, thereafter, each frame is captured at 0.5 second intervals towards the end of the simulation. Videos for each can be found in the Digital Appendix.

Figure 41 shows the best performing soft robots from five independent run using fitness search, but with domain randomisation implemented during the evolution. Once again, there are no noticeable differences between the morphologies and behaviours compared to previous results. Similar to previous results at this scale and resolution, all of the best performing soft robots use a jumping motion. The soft robots evolved in seeds 1, 3 and 4 all shift its momentum forward during the jumping motion using a groove-like structure as seen in previous results. The soft robot evolved in seed 3, contains a larger concave indentation, but has a similar effect on the jumping motion. A channelled structure is again observed in one of the results, seed 2, which resembles a Pneunet design. In this case, the channelled structure used by opposing actuators is the mechanism that drives the jumping soft robot. The channelled actuator at the back shifts the weight of the soft robot forward every time it lands, due to the angle of its channels and in turn actuating force, and the opposing channelled actuator at the front of

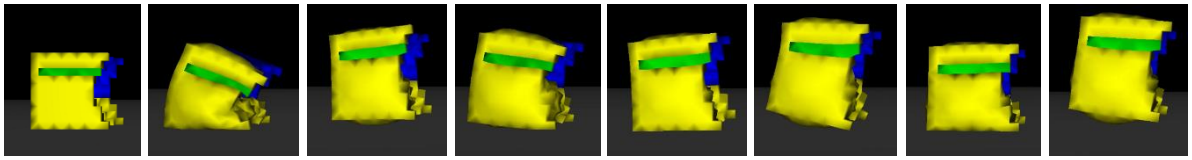
the robot pushes off the soft robot into the air. It also causes a small rotation such that the soft robot lands on actuator at the back, and the entire cycle is repeated.



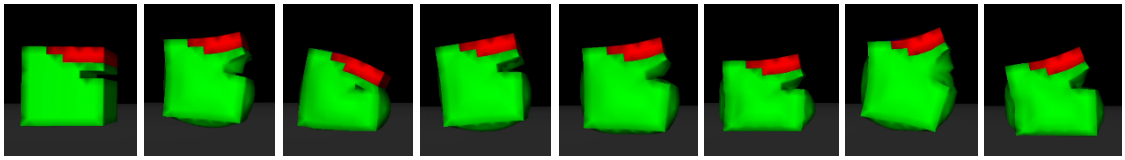
(a) Seed 1



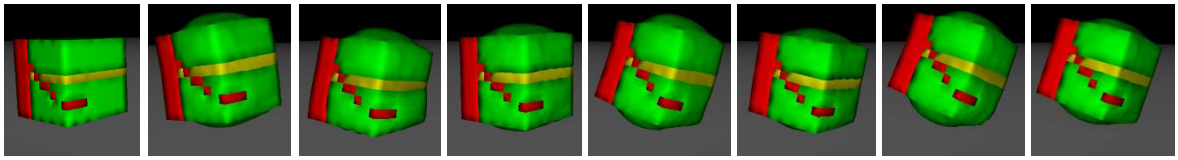
(b) Seed 2



(c) Seed 3



(d) Seed 4



(e) Seed 5

Figure 41: The motion of the top performing soft robots evolved from 5 independent runs using fitness search with domain randomisation. The first frame shows the initial shape, thereafter, each frame is captured at 0.5 second intervals towards the end of the simulation. Videos for each can be found in the Digital Appendix.

No observable difference was seen in the behaviour and structure between the three methods. Figure 42 also indicates that in terms of performance, as there is also no significant difference between the three methods. This contradicts previous literature, which has shown that novelty search outperforms fitness search in many cases. This contradiction can be attributed to two reasons. Firstly, as shown in Section 6.2, the modifications made to the NEAT settings significantly improved the exploration of the fitness

search method, this may have been a sufficient increase in exploration to compete with novelty search. Additionally, some of the best performing soft robots evolved have simple morphologies. This means that it is not necessary for more diversity to find these good performing solutions. However, these may be a local maximum, and given more generations novelty search should be more capable of finding more complex solutions that perform better. Despite the non-significant difference between the median performance, it should be noted that both the minimum and maximum performance achieved by novelty search was higher, which may indicate that although the medians are similar, novelty search will obtain these performances more often with a minimum performance close to that of the median. The inclusion of domain randomisation slight reduces the performance of fitness search. This is expected since the randomisation would make it more challenging for the evolution to converge on a result. Interestingly, the confidence interval of the performance drops in the last forty generations. This drop indicates that at that some point in one of the evolutionary runs, the best performing solution underperformed in one of the randomised environments such that it was not selected for the next generation. This is what allows domain randomisation to potentially overcome local maxima to find more general solutions that can perform in any environment.

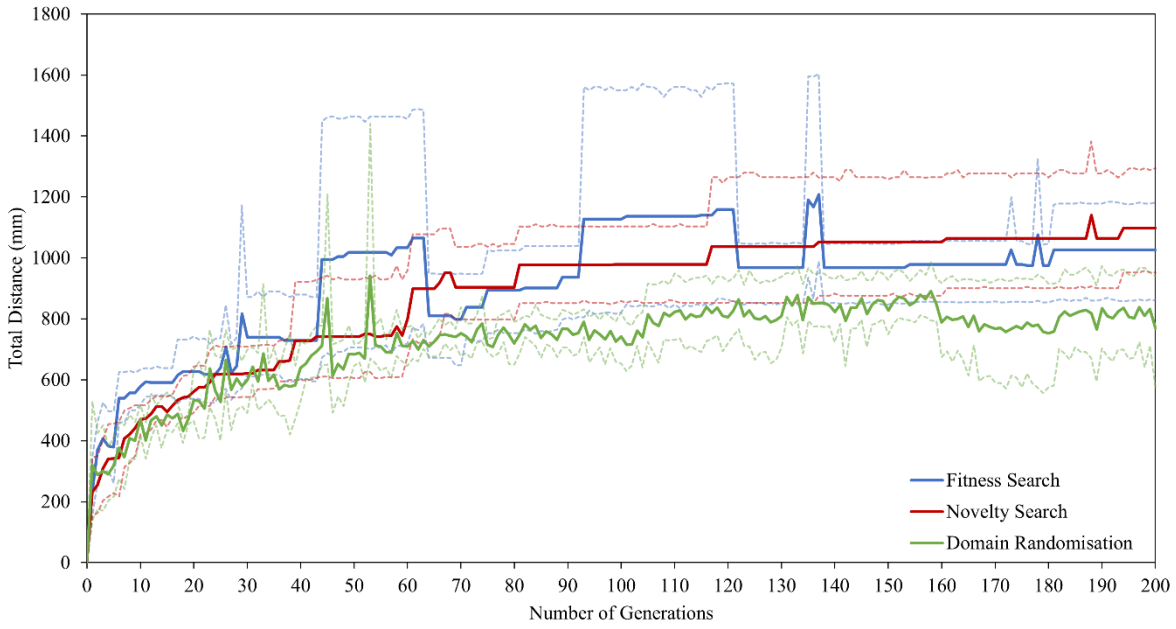


Figure 42: The median distance travelled by the top-performing soft robots. The solid line represents the median value, and the dotted lines represent the 95% bootstrapped confidence intervals.

Although the use of domain randomisation reduces the performance of the evolution slightly, domain randomisation encourages more robust generalised solutions that can adapt to changes in the environment. To test this, three out of the top ten performing soft robots from each run for each method were tested in two unseen environments of varying levels of difficulty. The three soft robots consisted of the best performing soft robot, and two other soft robots from the top ten that were unique in design

compared to the best soft robot and each other. This was done to increase the sample size whilst ensuring the soft robots were sufficiently different from each other. The two environments consisted of a sloped environment and a bumpy environment, as discussed in Section 5.4. Before comparing the different methods, it is worth noting the performance of the all the soft robots in these unseen environments. A median performance of 79.21% [62.50-89.58] was achieved on sloped surfaces and 89.92% [64.15-97.98] was achieved on the bumpy surface. This highlights how well soft robots can adapt to variations in their working environment. This is particularly interesting considering that all the evolved soft robots used a jumping motion, which would be expected to be unreliable on uneven terrain without using a closed loop control system to adjust for the terrain. However, because the soft robots are composed mostly out of soft material, the soft material can deform locally to accommodate the uneven terrain such that the effect on the overall motion of the soft robot is minimised.

Figure 43 shows the distribution of the performance of the soft robots on the sloped surface. Using the Wilcoxon's rank sum test, it was found that domain randomisation performs significantly better than fitness search in the slope environment for m=0.001 and m=0.002 (p=0.002 and p=0.009, respectively). Additionally, domain randomisation performs better than novelty search in the sloped environment for m=0.002 (p=0.036). Although the remaining results are non-significant, domain randomisation performed better based on their ranked sum than both novelty search and fitness search in all the sloped environments, except with m=0.003 compared to fitness search where fitness search performed slightly better.

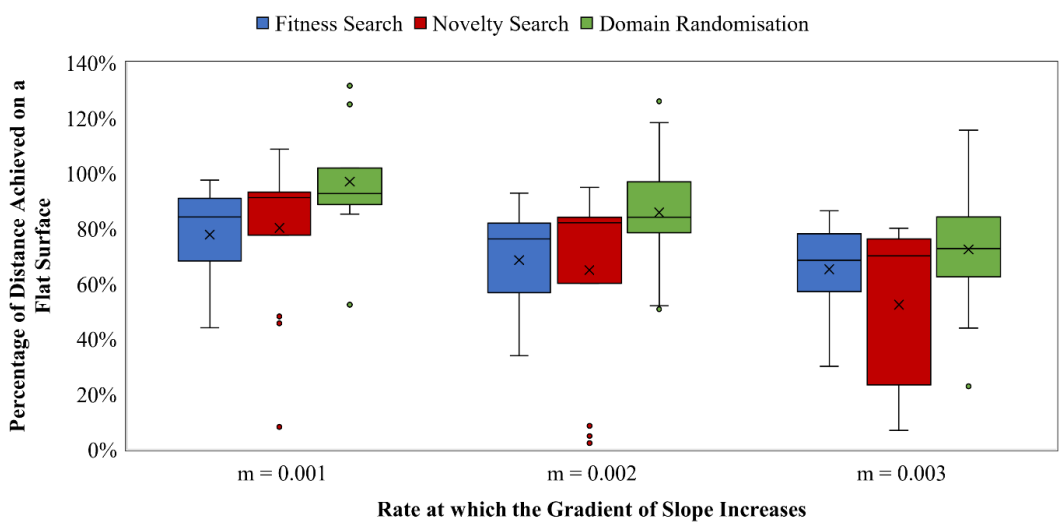


Figure 43: The performance of the top performing soft robots from each search method on a sloping surface. The surface is constructed using the formula: $y = mr^2$.

Figure 44 shows the p-values for the test, and as can be seen some of the results are very close to being significant, thus with larger sample sizes, these results may be more significant, further confirming how domain randomisation can improve the ability of NEAT to find more general and robust soft robots.

Figure 44 shows the distribution of the performance of the soft robots on the bumpy surface. Using the Wilcoxon's rank sum test, it was found that fitness search performs significantly better than novelty search where ω=1.5 and where ω=1.0 (p=0.010 and p=0.009, respectively). Similarly, fitness search with domain randomisation also performs significantly better than novelty search where ω=1.5 and where ω=1.0 (p=0.002 and p=0.001, respectively). This indicates that novelty search overfitted for the flat surface and failed to discover soft robots that can handle a variety of environments compared both fitness search and fitness search with domain randomisation. Surprisingly, although domain randomisation does appear to improve the performance of the fitness search at finding soft robots that can better handle different environments, no significance was found for the bumpy surface.

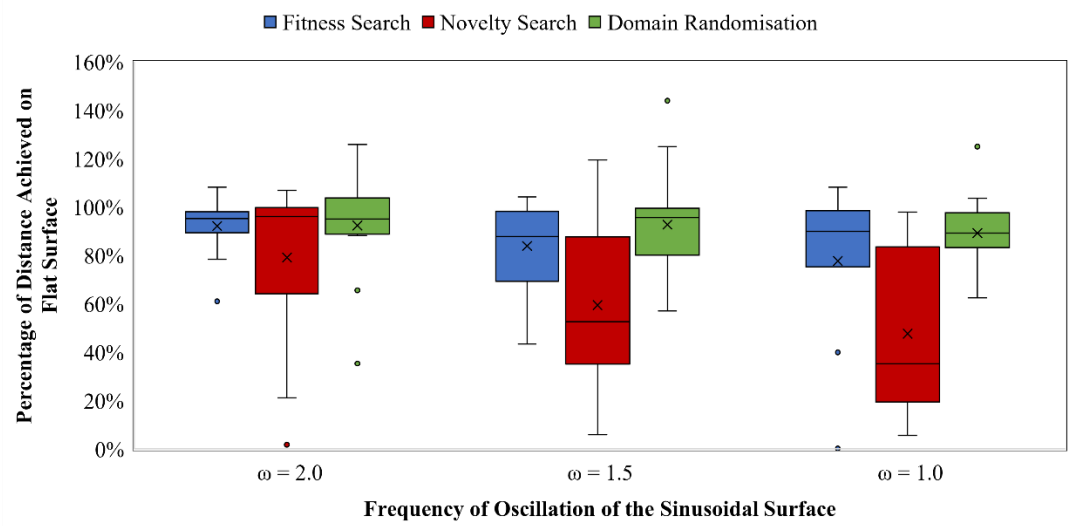


Figure 44: The performance of the top performing soft robots from each search method on a bumpy surface. The surface is constructed using the formula: $y = 0.1 \sin \omega r$.

Two out of the five best performing soft robots evolved using fitness search fell over when jumping along the flat surface. Thus, if the same occurs when being tested on other terrains their relative performance is the same. Conversely, if other robots that did not fall over on the flat terrain do when being tested on the unseen terrains, they will perform worse relative to their distance achieved on the flat terrain. For this reason, these soft robots may skew the data slightly, making it seem as though fitness search is discovering more general solutions, similarly to domain randomisation.

Domain randomisation has been shown to improve the ability of reinforcement learning algorithms at finding more general solutions that can handle variations, but also improve sim-to-sim and sim-to-real transfer. However, this is the first time it has been shown to also help evolutionary algorithms, particularly evolutionary robotics, at finding solutions that can handle variations. However, it is important to note that no conclusions can be made from these results on whether the ability to handle variations would translate to improvement in the real world. Additionally, it was hypothesised that domain randomisation could also improve the diversity of soft robots produced and the exploration of

the solution space. This was hypothesised since the environment is constantly changing in the natural world, requiring species to be sufficiently diverse to adapt to these changes from generation to generation. As stated in Section 5.4, the solution space could be quantified using properties of the soft robots such as material composition and actuator positions. This makes it possible to measure the amount exploration by analysing how many different material compositions or actuator positions were used during the evolutionary process. Figure 45 shows the number of unique material compositions and actuator positions discovered for each search method. As can be seen, the median exploration for novelty search seems to outperform both fitness search with and without domain randomisation; however, only significant improvement in exploration was found for the actuator 1 position between novelty search and fitness search ($p=0.038$, Wilcoxon's rank sum test). In contrast to what was hypothesised, domain randomisation slightly reduces the exploration of the solution space, but no significant difference was found.

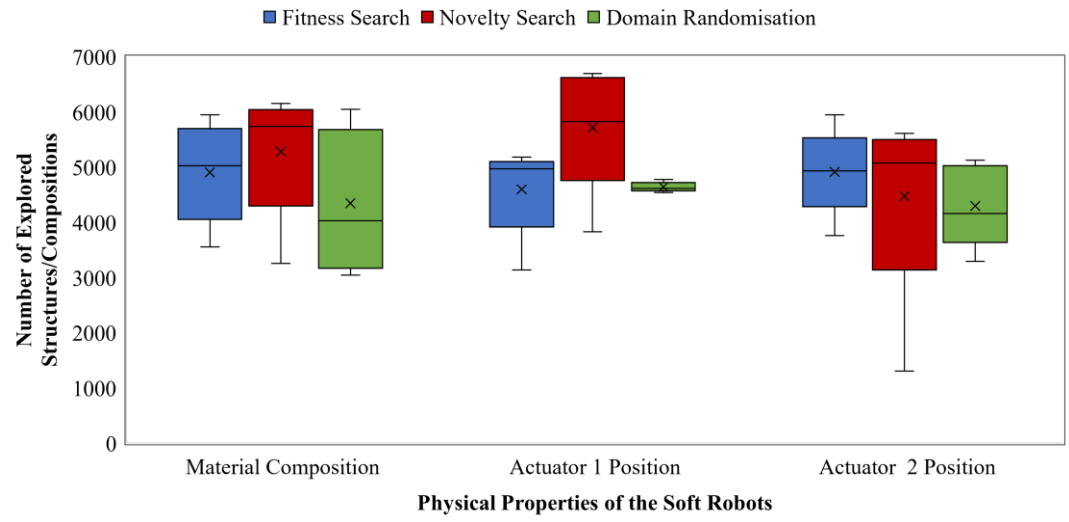


Figure 45: Material Composition: (a) Fitness Search (b) Novelty Search (c) Domain Randomisation

Figure 46 shows the median exploration of material compositions for fitness search (Figure 46a), novelty search (Figure 46b) and fitness search with domain randomisation (Figure 46c). All other heatmaps can be found in the digital appendix for the other runs can. The heatmaps indicate interesting similarities between the methods that give insight into what material compositions favoured due to convergence of the evolution. Most notably is that soft robots that utilise all the voxels available seem to be perform better and thus occur more frequently. This is expected, as soft robots that are bigger in size will be capable of producing larger actuation deformations and forces resulting in greater distances. For similar reasoning, soft robots evolved tend to have more actuating material than passive material. Lastly, there appears to be larger concentration of points along diagonal lines periodically in the heatmap. This is due to how simple CPPNs generate the soft robots. Simple CPPNs will commonly produce soft robots where the composition of split by a flat plane. For instance, a soft robot with a mesh

size of 10^3 may be split such that 500 elements are active and 500 elements are passive, or 600 elements are active and 400 elements are passive, and so on. This results in the diagonal lines observed in the heatmaps.

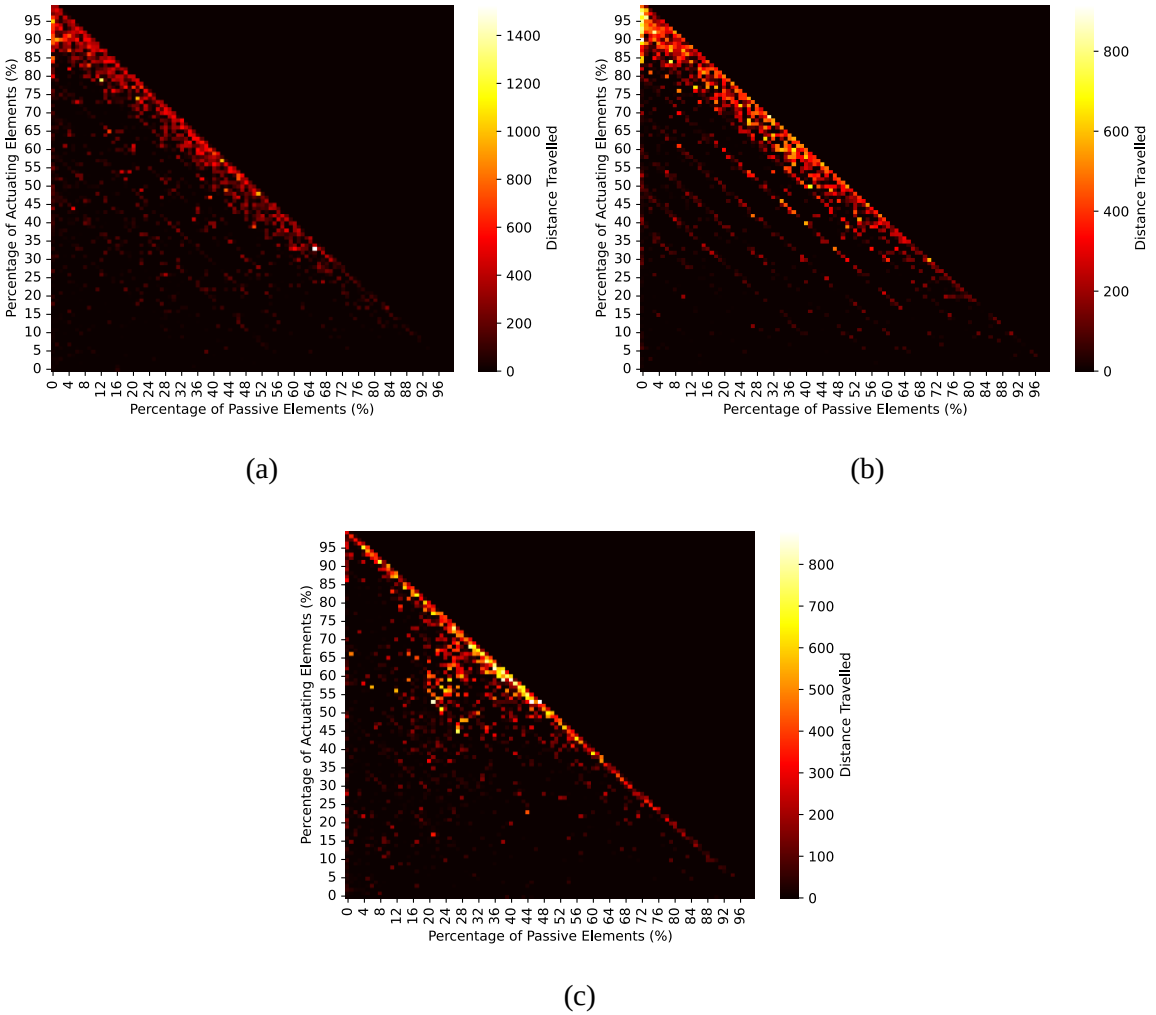
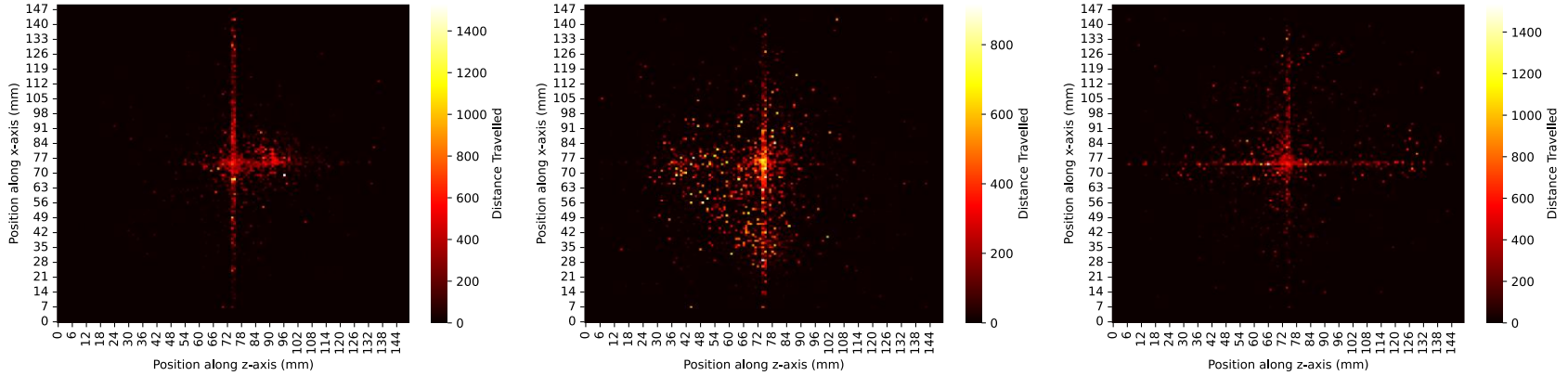


Figure 46: Heatmaps showing the median exploration of material compositions for each method: (a) fitness search, (b) novelty search, and (c) domain randomisation.

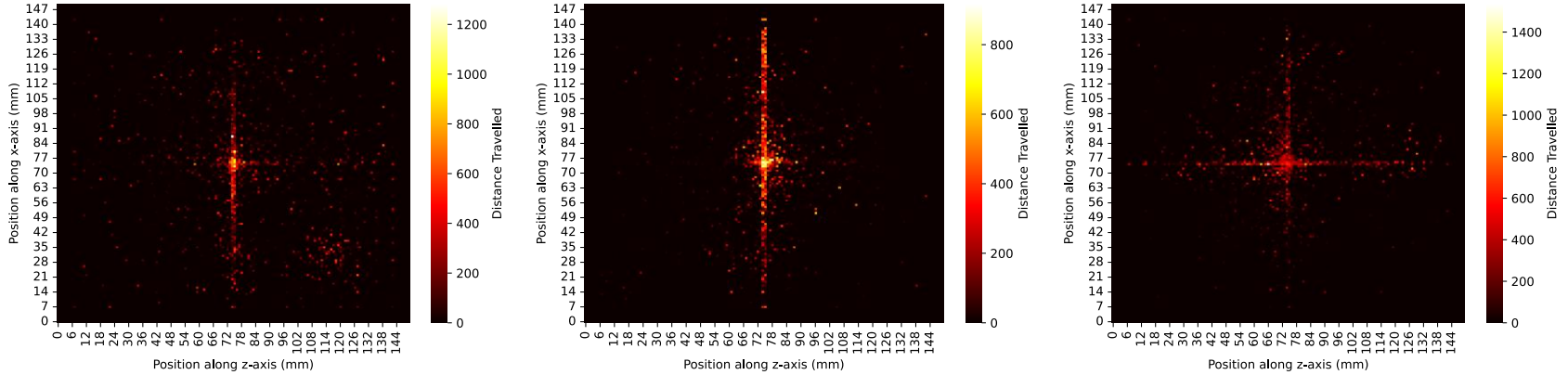
Figures 47 and 48 show the heatmaps of the actuator 1 and actuator 2 positions viewed from above, respectively, for the runs with median amount of exploration. Heatmaps showing the actuator positions from other views are provided in the digital appendix. As stated previously, simple CPPNs commonly generate soft robots which is divided into different materials across a flat plane. This is results in the cross-shaped heatmaps where centroid of the actuators are most frequently placed along a single axis to propel the soft robot in the corresponding direction. Another interesting observation can be seen in Figure AX in the digital appendix, where the actuator positions in the lower half of the soft robots tend to be converged around the central axis whereas actuator situated in the upper half of the robot are more

randomly distributed. This is because actuators in the lower half have a larger impact on how the soft robot moves, whereas actuators near the top have less effect on the movement. For this reason, actuators in the lower half are more frequently symmetrical around a central axis for stability during contact and to propel the soft robot in a particular direction along a straight path.



(a) (b) (c)

Figure 47: Heatmaps showing the median exploration of actuator 1 positions for each method: (a) fitness search, (b) novelty search, and (c) domain randomisation.



(a) (b) (c)

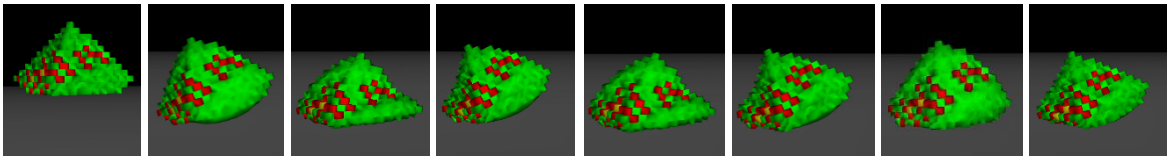
Figure 48: Heatmaps showing the median exploration of actuator 2 positions for each method: (a) fitness search, (b) novelty search, and (c) domain randomisation

6.3 Increasing the Solution Space Through Mesh Growth

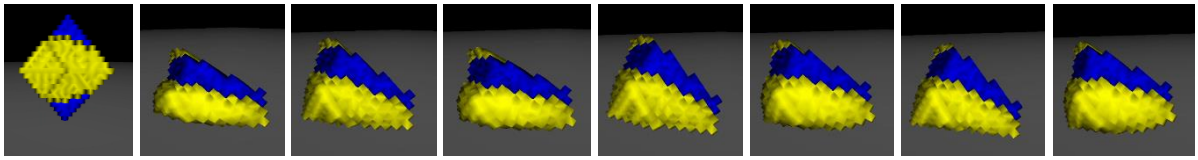
One of the proposed modifications to current methods of evolving soft robots was how positional information fed into the CPPNs to generate the shape of the soft robots. In previous research, a three-dimensional lattice in the shape of a cube was used. It was hypothesised that this was not only limiting the shape of the soft robot, but also the performance, and such a mesh growth method was proposed as a better alternative. Figure 49 shows the top performing soft robots from five independent runs using the mesh growth method. All the top performing robots resemble a pyramid shape with slight variations to this shape. This shape is very similar to the simplest shape that can be created with this method, since if the output of the CPPN that specified with material is present at a specific location is set at 1, the CPPN will generate a diamond shape using growth mesh method. The evolved soft robots in Figure 53 may indicate that the method may be biased towards this shape. However, an important feature that was consistently evolved was a flat underside to provide the stability required to perform well. Despite the close resemblance to the simplest shape generated by this method, the shape does aid in consistent movement. With an actuator on the underside of the soft robot, the diamond shape ensures that more inflation occurs at the centre of the robots. Additionally, the pyramid shape results in a high bending stiffness in the centre and low bending stiffnesses at the front, back and sides. Since these edges have lower bending stiffnesses they can bend to stabilise the soft robot if it lands on its underside slightly skew. For the above-mentioned reasons, the shapes of the evolved soft robots may also be attributed to performance not just a bias from the method. As with previous results, all the evolved soft robots used jumping as their locomotion method.

Seed 1 achieved this using a single actuator, which formed the main component of the soft robot. Additionally soft material was distributed along the length of its body but was unbalanced such that more weight was placed near rear end of its structure. The soft material also was shaped such as direct the momentum forward when the soft robot contacted the ground. The soft robot evolved in seed 2, falls over during the initialisation period during the simulation, such that it lands on one of its flat faces. In this position the rigid part of it, not only places the required weight on the actuator beneath to obtain a large actuation recoil but is also shaped as shift the momentum forward during the jumping locomotion. The soft robot in seed 3 uses both actuators during its motion. It consists of a rigid tail-like structure which pushed the soft robot forward onto its front actuator every time the soft robot contacts the ground. The front actuator propels the soft robot in the air, and a channelled opposing actuator pushes the actuator into the ground such that when it inflates, the deformation and returning force is larger. The evolved soft robot in seed 4, uses one actuator at near the back of its body to jump the robot forward. This results a slight rotation such that it lands on the front tip of the soft robot. This front tip is composed of the opposing actuator which has channels which gives it flexibility and a resisting force to stabilise the robot back onto the actuator at the back. Lastly, the evolved soft robot in seed 5 consists of soft material above actuating material. The actuator contains more volume near front. This asymmetry

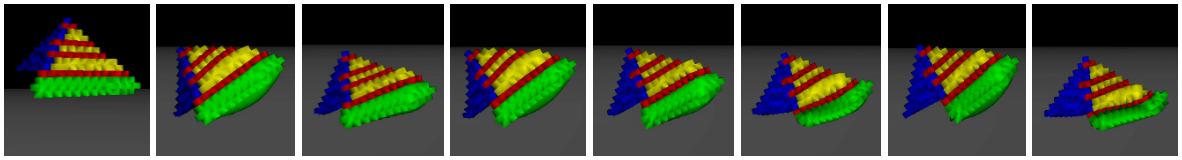
results in the soft robot jumping forward. Interestingly, in this run the evolution enhances the stability provided by the tips of the structure by elongating the tip at the back, making it function like a tail.



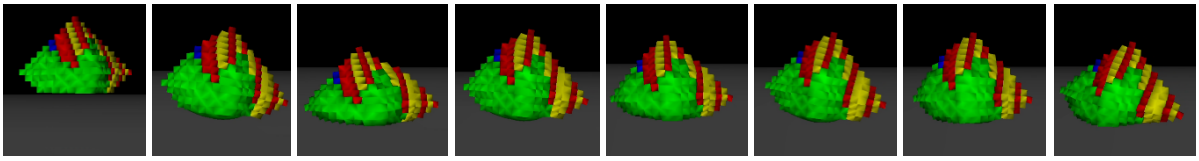
(a) Seed 1



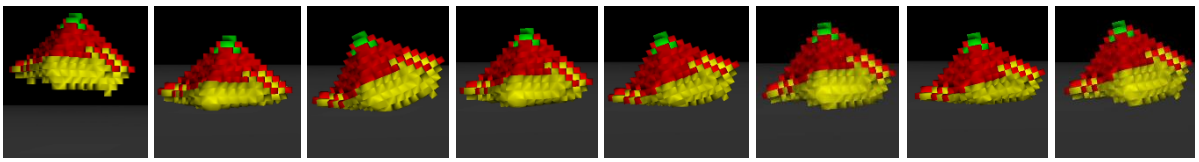
(b) Seed 3



(c) Seed 3



(d) Seed 4



(e) Seed 5

Figure 49: The motion of the top performing soft robots evolved from 5 independent runs using the growing mesh method. The first frame shows the initial shape, thereafter, each frame is captured at 0.5 second intervals towards the end of the simulation. Videos for each can be found in the Digital Appendix.

Figure 50, 51 and 52 shows the distribution of the length, width and height of all soft robots evolved during all five independent runs for both methods. It is substantially clear that the grid method has a strong bias towards the exact size of the input grid (10^3). Interestingly, is that the dimensions of the soft robots evolved using the growing mesh method are a lot more evenly distributed. This also confirms that the pyramid shape seen in the top-performing soft robots is not due to any bias of the method but rather it is a favourable shape for soft robots to perform well, since the evolution did explore other shapes of varying sizes. Although not shown in Figure 50, 51 and 52 for clarity, some soft robots were

evolved with dimensions up to 1000 voxels showing a wide range of soft robots that can be produced using this method.

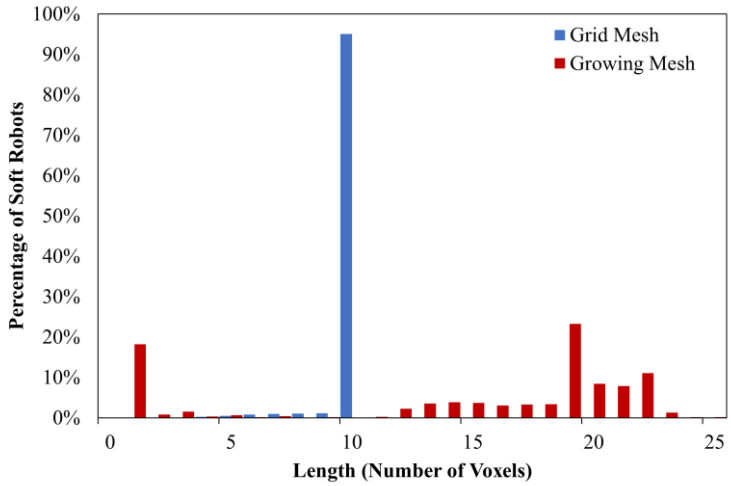


Figure 50: The frequency distribution of the length of soft robots from 5 evolutionary runs.

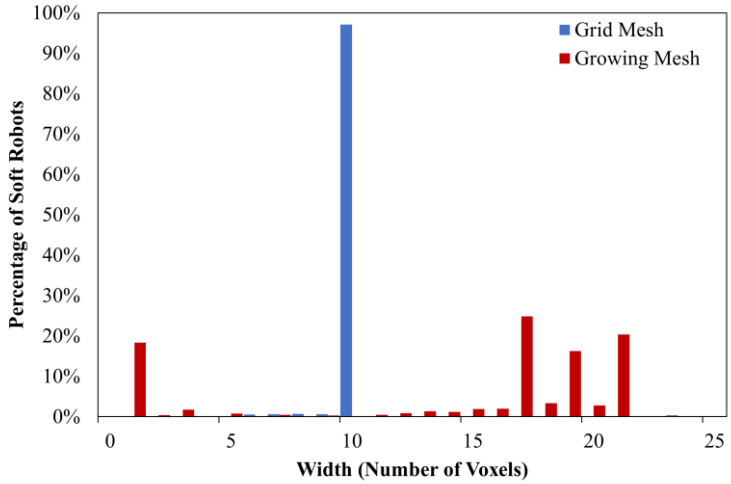


Figure 51: The frequency distribution of the width of soft robots from 5 evolutionary runs.

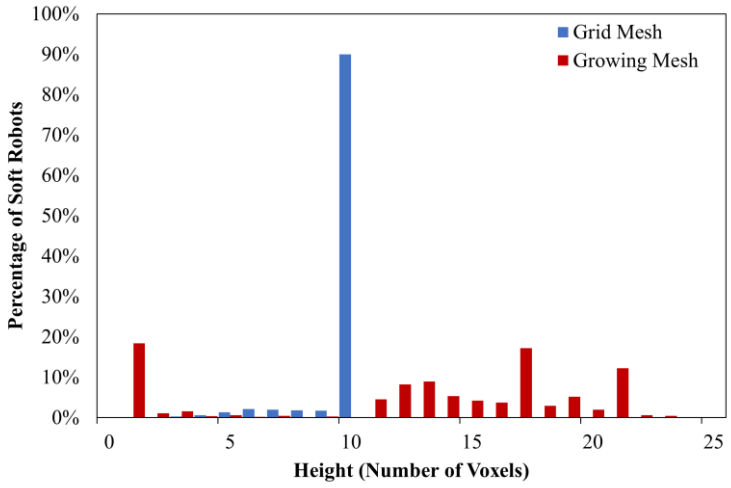


Figure 52: The frequency distribution of the height of soft robots from 5 evolutionary runs.

The proposed growing mesh method drastically improves that variety of soft robots that can be produced, but this could affect the performance as it increases the size of the solution space. Figure 53 shows the median performance of the top performing soft robots from five independent runs using both methods. Although the grid mesh method performs significantly better early on the evolution (at generation 80: $p=0.038$ using the Wilcoxon's rank sum test), growing mesh method performs better at generation 200; however, there is no significant difference ($p=0.125$). There are two reasons as to why the grid mesh method performs significantly better in the early stages of the evolution. The first reason is the increase in the solution space, requiring more time to explore the space and converge on a solution. Secondly, the simplest CPPNs will generate a cube when using the grid mesh method, which ensures that the soft robot has flat surface to rest on. In contrast, the simplest CPPNs will generate a diamond shape when using the growing mesh method, which does not have flat bottom, resulting in the soft robot falling over. Therefore, the evolution is required to grow the CPPN to generate the required flat bottom for stable locomotion on surface, which takes additional generations before it can converge on a solution. Despite this disadvantage, the method still manages to surpass the fitness search, and given another 100 generations may achieve a significant improvement compared to the grid mesh method.

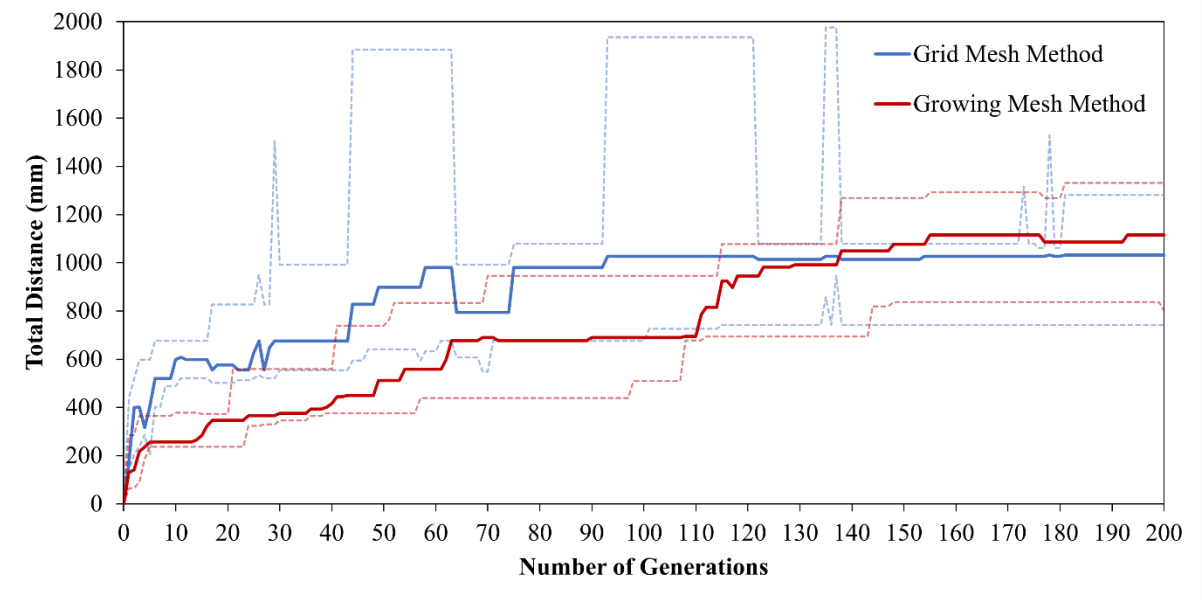
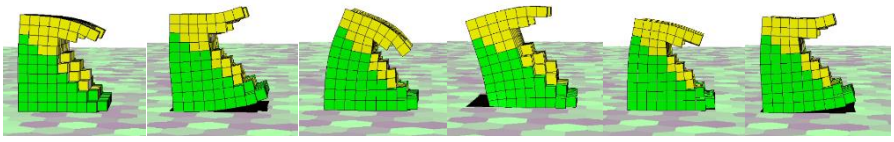


Figure 53: The median distance travelled by the top-performing soft robots. The solid line represents the median value, and the dotted lines represent the 95% bootstrapped confidence intervals.

6.4 Comparison between VoxCad and SOFA

Figure 54 shows three soft robots that were evolved in the VoxCad environment as well as their behaviour simulated in the SOFA simulation environment. Although it is unclear in Figure 54, all the soft robots failed to move across the surface, even though their actuators inflate as intended. The primary reason this happens lies within the actuation method in the two environments. In the VoxCad environment, each voxel increases in volume by 20% of its initial volume, thus each voxel deforms the exact same amount. In contrast, in SOFA the actuators also increase by 20% of its original volume but this is induced using an internal pressure inside the whole chamber. This results deformation of the actuator more in some places than others based on the local stiffness and resistance to the deformation. Since the actuator is in contact with the ground most of the deformation and force generated by the actuation is directed towards the sides and the top of the robot. The soft robot in turn is unable to generate enough force to push off the ground to move forward.



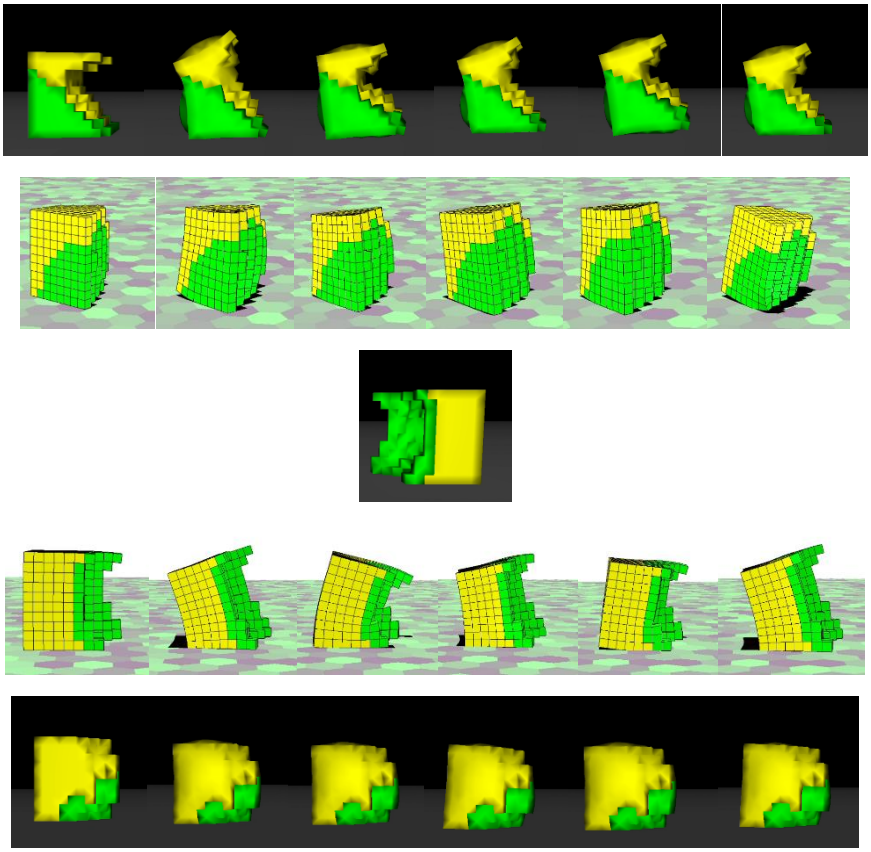


Figure 54: Comparison in the behaviour of soft robots evolved in VoxCad inside VoxCad versus SOFA. Videos for each can be found in the Digital Appendix.

Interestingly, is the lack of structures that resemble PneuNet actuators, which is also the case in previous work. This due to the same reason discussed above, and is even more prominent in these structures. PneuNet actuators are designed such that there is less resistance to deformation in one direction as well as a larger surface area to generate force in the same direction. This results in larger forces and motions in the designed direction. Since the actuation in VoxCad is less effected by the local stiffness of the structure, there is no need for PneuNet structures as they provide no advantages.

CONCLUSIONS

This research successively showed for the first time that soft robots that utilise fluidic elastomer actuators can be evolved in SOFA to move across a flat terrain. As a result, morphological structures that contain designs resembling PneuNet actuators was observed, which has not been observed in previous literature before. This resemblance between evolved soft robots in this work with real soft robots, indicates the realism of the set-up used.

Modifications made to the NEAT settings and to the fitness function failed to improve the performance of the evolution. However, importantly, the modified NEAT settings significantly improved the number of explored CPPN structures during the evolution. The modified fitness function successively evolved soft robots with consistent periodic motion that move along a straight path. In contrast, using total distance travelled by the soft robot as its fitness, resulted in 1 out of 5 runs resulted in a soft robot that failed to move forward with periodic motion and 3 out of 5 runs resulting in soft robots that do not move along a straight path.

Domain randomisation successively aided in evolving soft robots that could better handle variations, noise, and unseen environments, which indicates the potential of its use not only in evolutionary robotics, but also other evolutionary strategies. Domain randomisation failed to improve the performance of the evolution of soft robots. Additionally, in opposition to what was hypothesised domain randomisation reduced the exploration of the solution space. However, it was found that this was partly because of its effect on the stagnation algorithm used in CPPN-NEAT.

It was shown that the use of the grid methods results in a significant bias in the shape and size of the soft robots produced, whereas growing soft robots from a central element was shown to produce soft robots with a wider distribution of shapes and sizes. Due to the increase in the solution space, using the growing mesh method resulted in significantly worse performance of the evolution in early stages of the evolution (at generation 80). However, after 200 generations there is no significant difference in the performance between the two methods. This indicates that the growing mesh method requires more generations to converge on the same level of performance.

It was shown that soft robots evolved in VoxCad behave very differently in SOFA and in turn underperform in this more realistic environment. It is also shown as to why structures resembling PneuNets are unlikely to be evolved in VoxCad yet can be advantageous in SOFA.

7 RECOMMENDATIONS FOR FUTURE RESEARCH

Although this research has successfully evolved soft robots that use fluidic elastomer actuators to move across a flat terrain, there is more research required to improve the current methods and further make evolved soft robots have real applicability. The recommendations for future work are as follows:

- Modify the stagnation algorithm of NEAT such that it is unaffected by domain randomisation, whilst maintaining its ability to perform its required function.
- A significant improvement required before evolved soft robots can be used for real applications, is the implementation of self-collision without excessively slowing down the evolutionary computation.
- The most dominant evolved behaviour was jumping, which logically works when using inflating actuators, thus, further research should be performed on the use of vacuum-powered and/or DEAs. These actuators behave more closely to natural muscles, thus more interesting and natural behaviour could be found. Additionally, these actuators have better power-to-weight ratios for more challenging tasks.
- This research should be repeated for different tasks and environments to provide more insight into the methods.
- Lastly, our understanding of natural evolution and developmental biology is ever-changing. For example, in a recent study, it was shown that mutations are not completely random and are different throughout our genome. These discoveries can help improve CPPN-NEAT to improve the performance of evolving soft robots using evolutionary strategies.

REFERNCES

[1] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson, “Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding,” in *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference*, 2013, doi: 10.1145/2463372.2463404.

[2] M. Joachimczak, R. Suzuki, and T. Arita, “Fine Grained Artificial Development for Body-Controller Coevolution of Soft-Bodied Animats,” 2014, doi: 10.7551/978-0-262-32621-6-ch040.

[3] D. Vokoun, J. Pilch, L. Kadeřávek, and P. Šittner, “Strength of superelastic niti velcro-like fasteners,” *Metals (Basel).*, vol. 11, no. 6, Jun. 2021, doi: 10.3390/MET11060909.

[4] Y. F. Fu, C. Q. Yuan, and X. Q. Bai, “Marine drag reduction of shark skin inspired riblet surfaces,” *Biosurface and Biotribology*, vol. 3, no. 1, pp. 11–24, Mar. 2017, doi: 10.1016/J.BSBT.2017.02.001.

[5] C. T. Foo, B. Omar, and I. Taib, “Shape Optimization of High-Speed Rail by Biomimetic,” *MATEC Web Conf.*, vol. 135, Nov. 2017, doi: 10.1051/MATECCONF/201713500019.

[6] N. Savage, “How AI and neuroscience drive each other forwards,” *Nature*, vol. 571, no. 7766, pp. S15–S17, Jul. 2019, doi: 10.1038/D41586-019-02212-4.

[7] Z. Michalewicz and M. Schoenauer, “Evolutionary Algorithms,” *Encycl. Inf. Syst.*, pp. 259–267, Jan. 2003, doi: 10.1016/B0-12-227240-4/00065-4.

[8] T. R. Gregory, “Understanding Natural Selection: Essential Concepts and Common Misconceptions,” *Evol. Educ. Outreach*, vol. 2, no. 2, pp. 156–175, May 2009, doi: 10.1007/S12052-009-0128-1/FIGURES/3.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016, doi: 10.1109/CVPR.2016.91.

[10] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” pp. 1–14, Sep. 2014.

[11] M. C. Kenton, L. Kristina, and J. Devlin, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” no. Mlm, 1953.

[12] T. B. Brown *et al.*, “Language models are few-shot learners,” *Adv. Neural Inf. Process. Syst.*, vol. 2020-Decem, 2020.

[13] Z. Yang, Z. Dai, Y. Yang, and J. Carbonell, “XLNet : Generalized Autoregressive Pretraining for Language Understanding,” no. NeurIPS, pp. 1–18, 2019.

[14] OpenAI *et al.*, “Learning Dexterous In-Hand Manipulation,” *Int. J. Rob. Res.*, vol. 39, no. 1, pp. 3–20, Aug. 2018.

[15] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” *Evol. Comput.*, 2002, doi: 10.1162/106365602320169811.

[16] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989, doi: 10.1016/0893-6080(89)90020-8.

[17] C. Debao, “Degree of approximation by superpositions of a sigmoidal function,” *Approx. Theory its Appl.*, vol. 9, no. 3, pp. 17–28, 1993, doi: 10.1007/BF02836480.

[18] K. Sims, “Evolving virtual creatures,” in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1994*, 1994, doi: 10.1145/192161.192167.

[19] “About the Human Genome Project.” [Online]. Available: https://web.ornl.gov/sci/techresources/Human_Genome/project/index.shtml. [Accessed: 31-Mar-2022].

[20] G. Methenitis, “Evolution of Soft Robots by Novelty Search,” no. November, 2014.

[21] K. Sims, “Evolving 3D Morphology and Behavior by Competition,” *Artif. Life*, 1994, doi: 10.1162/artl.1994.1.4.353.

[22] P. Eggenberger-Hotz, “Evolving Morphologies of Simulated 3d Organisms Based on Differential Gene Expression,” *Proc. 4th Eur. Conf. Artif. Life*, 1997, doi: 10.1007/s13398-014-0173-7.2.

[23] M. Joachimczak, R. Suzuki, and T. Arita, “Improving evolvability of morphologies and controllers of developmental soft-bodied robots with novelty search,” *Front. Robot. AI*, 2015, doi: 10.3389/frobt.2015.00033.

[24] M. Joachimczak, T. Kowaliw, R. Doursat, and B. Wróbel, “Brainless bodies: Controlling the development and behavior of multicellular animats by gene regulation and diffusive signals,” in *Artificial Life 13: Proceedings of the 13th International Conference on the Simulation and Synthesis of Living Systems, ALIFE 2012*, 2012, doi: 10.7551/978-0-262-31050-5-ch046.

[25] M. Joachimczak, R. Suzuki, and T. Arita, “Fine grained artificial development for body-controller coevolution of soft-bodied animats,” *Artif. Life 14 - Proc. 14th Int. Conf. Synth. Simul.*

Living Syst. ALIFE 2014, pp. 239–246, 2014, doi: 10.7551/978-0-262-32621-6-ch040.

[26] K. O. Stanley, "Compositional pattern producing networks: A novel abstraction of development," *Genet. Program. Evolvable Mach.*, 2007, doi: 10.1007/s10710-007-9028-8.

[27] F. Corucci, N. Cheney, H. Lipson, C. Laschi, and J. C. Bongard, "Evolving swimming soft-bodied creatures," *Late Break. Proc. Fifteenth Int. Conf. Synth. Simul. Living Syst. (ALIFE XV)*, no. January, p. 6, 2016.

[28] J. E. Auerbach and J. C. Bongard, "Evolving complete robots with CPPN-NEAT: The utility of recurrent connections," *Genet. Evol. Comput. Conf. GECCO'11*, no. January, pp. 1475–1482, 2011, doi: 10.1145/2001576.2001775.

[29] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Front. Robot. AI*, vol. 3, no. JUL, pp. 1–17, 2016, doi: 10.3389/frobt.2016.00040.

[30] J. Lehman and K. O. Stanley, "Evolving a Diversity of Creatures through Novelty Search and Local Competition," 2011.

[31] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," Apr. 2015, doi: 10.48550/arxiv.1504.04909.

[32] A. Gaier, A. Asteroth, and J. B. Mouret, "Are quality diversity algorithms better at generating stepping stones than objective-based search?," *GECCO 2019 Companion - Proc. 2019 Genet. Evol. Comput. Conf. Companion*, no. September, pp. 115–116, 2019, doi: 10.1145/3319619.3321897.

[33] J. Nordmoen, F. Veenstra, K. O. Ellefsen, and K. Glette, "MAP-Elites Enables Powerful Stepping Stones and Diversity for Modular Robotics," *Front. Robot. AI*, vol. 8, p. 56, Apr. 2021, doi: 10.3389/FROBT.2021.639173/BIBTEX.

[34] A. Cully *et al.*, "Robots that can adapt like animals To cite this version : HAL Id : hal-01158243 Robots that can adapt like animals," vol. 521, no. 7553, pp. 503–507, 2015.

[35] G. Alici, "Softer is harder: What differentiates soft robotics from hard robotics?," in *MRS Advances*, 2018, vol. 3, no. 28, pp. 1557–1568, doi: 10.1557/adv.2018.159.

[36] N. El-Atab *et al.*, "Soft Actuators for Soft Robotic Applications: A Review," *Adv. Intell. Syst.*, vol. 2, no. 10, p. 2000128, 2020, doi: 10.1002/aisy.202000128.

[37] "Tutorial: making a soft gripper | Soft Robotics Toolkit." [Online]. Available: https://softroboticstoolkit.com/sofa/tutorial. [Accessed: 31-Mar-2022].

[38] F. Renda, M. Cianchetti, M. Giorelli, A. Arienti, and C. Laschi, "A 3D steady-state model of a tendon-driven continuum soft manipulator inspired by the octopus arm," *Bioinspiration and*

Biomimetics, vol. 7, no. 2, Jun. 2012, doi: 10.1088/1748-3182/7/2/025006.

[39] P. Boyraz, G. Runge, and A. Raatz, "An overview of novel actuators for soft robotics," *High-Throughput*. 2018, doi: 10.3390/act7030048.

[40] Christopher W. Lim, "UC Irvine UC Irvine Electronic Theses and Dissertations UNIVERSITY ! OF ! CALIFORNIA , !," pp. 1982–2004, 2015.

[41] F. Ilievski, A. D. Mazzeo, R. F. Shepherd, X. Chen, and G. M. Whitesides, "Chemical Robotics Soft Robotics for Chemists**," doi: 10.1002/anie.201006464.

[42] Y. Y. Lin, "Design of Pneumatic Soft Surgical Robot For Endoscopic NOTES and MIS Application with Reduced Size," p. 71, 2019.

[43] C. D. Onal, X. Chen, G. M. Whitesides, and D. Rus, "Soft mobile robots with on-board chemical pressure generation," *Springer Tracts Adv. Robot.*, vol. 100, pp. 525–540, 2017, doi: 10.1007/978-3-319-29363-9_30.

[44] M. Wehner *et al.*, "An integrated design and fabrication strategy for entirely soft, autonomous robots," *Nature*, vol. 536, no. 7617, pp. 451–455, 2016, doi: 10.1038/nature19100.

[45] D. Yang *et al.*, "Buckling Pneumatic Linear Actuators Inspired by Muscle," *Adv. Mater. Technol.*, vol. 1, no. 3, Jun. 2016, doi: 10.1002/ADMT.201600055/ABSTRACT.

[46] Z. Jiao, C. Zhang, W. Wang, M. Pan, H. Yang, and J. Zou, "Advanced Artificial Muscle for Flexible Material-Based Reconfigurable Soft Robots," *Adv. Sci.*, vol. 6, no. 21, Nov. 2019, doi: 10.1002/ADVS.201901371.

[47] M. Franke, A. Ehrenhofer, S. Lahiri, E. F. M. Henke, T. Wallmersperger, and A. Richter, "Dielectric Elastomer Actuator Driven Soft Robotic Structures With Bioinspired Skeletal and Muscular Reinforcement," *Front. Robot. AI*, vol. 7, no. December, pp. 1–11, 2020, doi: 10.3389/frobt.2020.510757.

[48] K. Jung, J. C. Koo, J. Do Nam, Y. K. Lee, and H. R. Choi, "Artificial annelid robot driven by soft actuators," *Bioinspiration and Biomimetics*, vol. 2, no. 2, p. 3182, 2007, doi: 10.1088/1748-3182/2/2/S05.

[49] F. Sadeghi and S. Levine, "CAD2RL: Real Single-Image Flight without a Single Real Image," *Robot. Sci. Syst.*, vol. 13, Nov. 2016, doi: 10.48550/arxiv.1611.04201.

[50] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World."

[51] J. E. Auerbach and J. C. Bongard, "On the relationship between environmental and morphological complexity in evolved robots," in *GECCO'12 - Proceedings of the 14th*

International Conference on Genetic and Evolutionary Computation, 2012, doi: 10.1145/2330163.2330238.

[52] M. Jelisavcic, K. Glette, E. Haasdijk, and A. E. Eiben, "Lamarckian Evolution of Simulated Modular Robots," *Front. Robot. AI*, 2019, doi: 10.3389/frobt.2019.00009.

[53] J. E. Auerbach *et al.*, "Robogen: Robot generation through artificial evolution," *Artif. Life 14 - Proc. 14th Int. Conf. Synth. Simul. Living Syst. ALIFE 2014*, pp. 136–137, 2014, doi: 10.7551/978-0-262-32621-6-CH022.

[54] J. Rieffel and S. Smith, "Growing and evolving soft robots with a face-encoding tetrahedral grammar," *GECCO'12 - Proc. 14th Int. Conf. Genet. Evol. Comput. Companion*, pp. 1457–1458, 2012, doi: 10.1145/2330784.2330988.

[55] "Undergraduates Hunt for Special Tetrahedra That Fit Together | Quanta Magazine." [Online]. Available: https://www.quantamagazine.org/mit-math-students-continue-aristotles-tetrahedra-tiling-20210209/. [Accessed: 31-Mar-2022].

[56] J. Hiller and H. Lipson, "Dynamic Simulation of Soft Multimaterial 3D-Printed Objects," *https://home.liebertpub.com/soro*, vol. 1, no. 1, pp. 88–101, Feb. 2014, doi: 10.1089/SORO.2013.0010.

[57] M. Joachimczak and B. Wróbel, "Evolution of the morphology and patterning of artificial embryos: Scaling the tricolour problem to the third dimension," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5777 LNAI, no. PART 1, pp. 35–43, 2011, doi: 10.1007/978-3-642-21283-3_5.

[58] L. Brodbeck, S. Hauser, and F. Iida, "Morphological evolution of physical robots through model-free phenotype development," *PLoS One*, 2015, doi: 10.1371/journal.pone.0128444.

[59] S. Whiteson, P. Stone, K. O. Stanley, R. Miikkulainen, and N. Kohl, "Automatic feature selection in neuroevolution," *GECCO 2005 - Genet. Evol. Comput. Conf.*, no. June, pp. 1225–1232, 2005, doi: 10.1145/1068009.1068210.