Evolving Soft Robots for Control Using a Gene Regulatory Network

By Nathaniel Brookes

21011025

BSc in Computer Science

26th April 2024

SCHOOL OF COMPUTER SCIENCE AND MATHEMATICS
Keele University
Keele
Staffordshire
ST5 5BG

Abstract

This project investigates the effectiveness of Gene Regulatory Networks (GRN) for controlling Voxel-based Soft Robots on locomotion and object manipulation tasks. GRNs are a biologically-inspired model that has been shown to be effective at controlling rigid robots; however, this model is under-researched when applied to soft robots.

An abstract recurrent-based GRN model is implemented in Python and applied within a soft robot simulator. Different experiments are conducted to compare the performance of evolved GRN-controlled robots against a baseline approach.

The first experiment studies the GRN's ability to control hand-designed robots. The GRN's initial performance was poor; it performed statistically worse than the baseline in every task. After improving the GRN model, it achieved comparable performance with the baseline approach and outperformed the baseline in one locomotion task.

The GRN model was adapted by implementing a developmental model and a decentralised controller, allowing the GRN to control the robot's shape and behaviour. The results show that the co-evolved GRN designs and controls effective robots for each task. However, the co-evolved GRN was not able to outperform the baseline approach.

Overall, the results demonstrate that this GRN model can effectively design and control autonomous soft robots for locomotion and object manipulation tasks.

Acknowledgements

I want to thank my supervisor, Dr Alastair Channon, for his guidance and Ash Leake for setting up a lab PC for running experiments.

Table of Contents

Table of Figures

Table of Appendix Figures

Specialist Glossary

GRN	Gene Regulatory Network
PPO	Proximal Policy Optimisation
AI	Artificial Intelligence
VSR	Voxel-based Soft Robot
ANN	Artificial Neural Network
PPO+GA	PPO used with a Genetic Algorithm
OOP	Object-Oriented Programming
UML	Unified Modelling Language
EA	Evolutionary Algorithm

Introduction

Problem Statement and Motivation

One aim for robotics is to develop machines that can operate independently without human supervision. This has important applications such as search-and-rescue missions and traversing the harsh terrain of another planet, tasks which are often too dangerous or infeasible for humans. (Wong et al., 2018) With this comes the challenge of finding an effective mechanism to control the robot.

Designing a robust controller by hand is difficult since it must adapt to new environmental conditions. (Patel et al., 2001). As such, Artificial Intelligence methods have been applied to optimise agent behaviour, including Artificial Neural Networks (Liu et al., 2023) and Reinforcement Learning (Singh, Kumar & Singh, 2022).

These approaches have traditionally been applied to rigid robots, such as in Karl Sims' seminal paper, where the morphology and controllers of 3D block-based robots are evolved (Sims, 1994). However, rigid robots are constrained by their fixed structure, which limits their abilities. (Kriegman et al., 2017) This also does not reflect biology, where organisms have soft components.

The advantages of soft robots, which can morph and adapt to their environment, have led researchers to investigate how AI can be applied to control soft robots. Recently, (Bhatia et al., 2021) used Proximal Policy Optimisation (PPO), a reinforcement learning algorithm, to optimise the controller of Voxel-based Soft Robots (VSR) to solve locomotion and object manipulation tasks. VSRs are modular robots with connected voxels that can individually expand and contract. (Hiller, Lipson, 2012)

Gene Regulatory Networks (GRN) are a biologically-inspired model based on the dynamic interactions within our cells. This model is under-researched and could provide an alternative method for controlling soft robots compared to conventional reinforcement learning.

Overview of the Project

This project will investigate the effectiveness of an evolved GRN model for controlling soft robots. The GRN model will be implemented in Python and applied to locomotion and object-manipulation tasks. The GRN's performance will be compared against PPO through experiments, and different approaches will be considered to improve the GRN's performance following initial results.

Research Question

Can GRNs be evolved to effectively control soft robots on locomotion and object manipulation tasks?

Aims	
1	Explore the effectiveness of Gene Regulatory Networks as a suitable model for controlling Voxel-Soft-bodied Robots across different simulated control tasks and environments.
2	Compare and evaluate the performance of robots evolved using a GRN model with a conventional model across various control tasks and study the capabilities and behaviours of each model.
3	Explore methods to improve the performance of the GRN model.
Objectives	
1	Replicate the findings of a relevant research paper where Voxel-based Soft Robots are controlled with a conventional model. This will act as the baseline model, which will be used for comparing against the GRN-inspired model.
2	Design and implement a GRN model, which can be evolved to control of a soft-bodied agent to achieve particular tasks.
3	Conduct experiments to compare the GRN with the existing baseline model on various agent control tasks.
4	Use relevant statistical methods to compare and analyse the evolutionary performance of the robots in each experiment.
5	Investigate how varying the GRN model's hyperparameters or using a different GRN model could improve the evolved robots' performance.
6	If the GRN can successfully control robots to complete a simple task, conduct further experiments on the GRN model to assess its ability to control robots on more complex control tasks and environments

Fig.1. *Project Aims and Objectives*

Chapter 1: Literature Review

1.1 Gene Regulatory Networks

Gene Regulatory Networks (GRN) originate from biology, where they are used to infer the complex interactions that occur within cells, as DNA is transcribed into mRNA and translated into proteins. Certain proteins regulate transcription by binding to the DNA, which inhibits or activates particular genes and dynamically affects the cell's behaviour. (Delgado, Gómez-Vela, 2019)

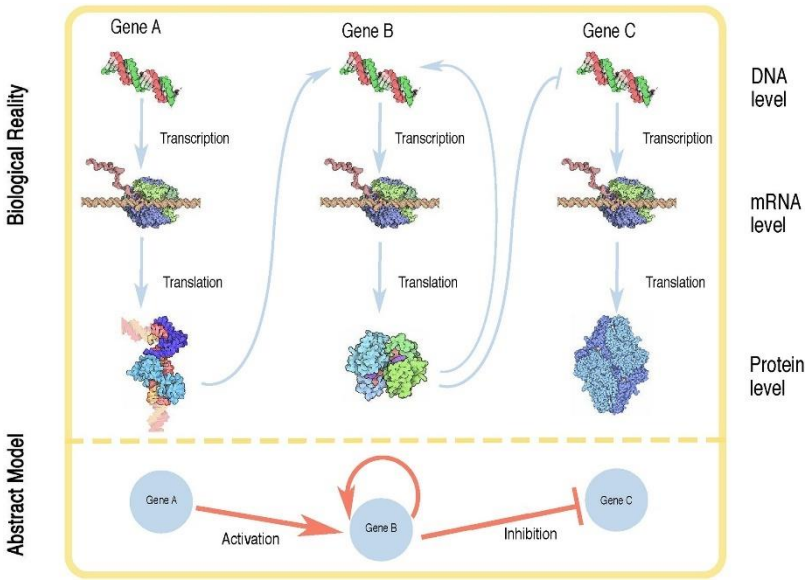


Fig.2. *Example of biological interactions and the inferred model (Delgado, Gómez-Vela, 2019)*

The dynamic nature of GRNs means they could be a good model for controlling soft-bodied agents. GRNs can have recurrent interactions, as shown in Fig.2, which could generate cyclical feedback loops for walking.

GRN models also exhibit memory. (Watson et al., 2014) showed that evolving an abstract GRN is equivalent to associative learning of weights in a Hopfield network,

and their experiments show that a GRN can be evolved to store and recall multiple patterns.

GRN models have successfully controlled artificial agents in various tasks and environments. (Cussat-Blanc, Harrington & Banzhaf, 2019)

1.1.1 Gene Regulatory Networks for Rigid Agent Control

A biologically-inspired cell-based GRN model was evolved using a genetic algorithm to control a 2D robot, where the goal for the robot is to traverse the space without colliding with obstacles. (Asr, Majd, 2013) Their model uses different genes to code for proteins, which can combine with other proteins. Proteins can regulate genes if their structure is similar to the gene's binding site. Their model successfully controls the robot using four genes.

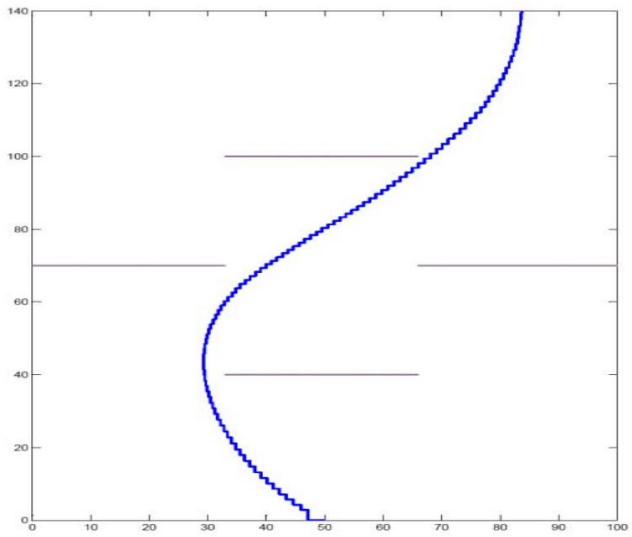


Fig.3. *Path showing the evolved robot navigating the obstacles (Asr, Majd, 2013)*

A similar cell-based GRN was evolved using a genetic algorithm to control a 3D modular snake robot. (Zahadat et al., 2013) Their model uses fractals to define the

proteins, and the robot was evaluated on how quickly it could move. This study suggests that their model can effectively control modular robots.

Another paper evolved a tag-based GRN to control a 2D virtual racecar. (Sanchez, Cussat-Blanc, 2014) Their GRN consists of abstract proteins composed of ID, enhancer, and inhibitor tags, and they have three types: input, regulatory, and output. This model is more abstract than the cell-based models, as some cell processes, such as protein-to-protein interactions, are not modelled.

This paper found that a GRN can control a simulated car and effectively handle cooperative and conflicting behaviour within the same network. However, they found that GRNs generate side effects in larger networks and produce inefficient solutions.

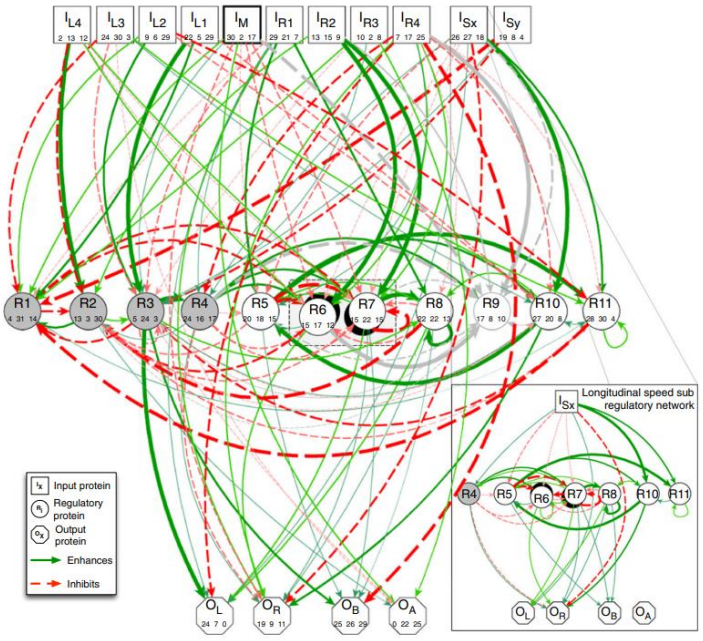


Fig.4. *Diagram of the GRN (Sanchez, Cussat-Blanc, 2014)*

A similar abstract GRN was used to control 2D robots in simple tasks such as reaching a light source. (Moreira, Rennó-Costa, 2021) Their model contains sensor,

processor and controller genes with recurrence connections. They found that larger GRNs are necessary for complex tasks; however, too many genes can introduce noise.

1.1.2 Gene Regulatory Networks for Soft Agent Control

While most GRNs have been applied to control rigid agents, they have also been applied to control soft-bodied agents:

A recurrent neural-network GRN was evolved to control 2D soft-bodied animats for locomotion. (Joachimczak, Suzuki & Arita, 2014) Interestingly, this GRN was also used to control the animat's shape via a developmental process, where each animat is grown from a single cell, which progressively divides to form a multicellular organism. This paper utilised the NEAT algorithm to evolve the animats' GRNs, producing diverse morphologies that exhibit animal-like gaits.

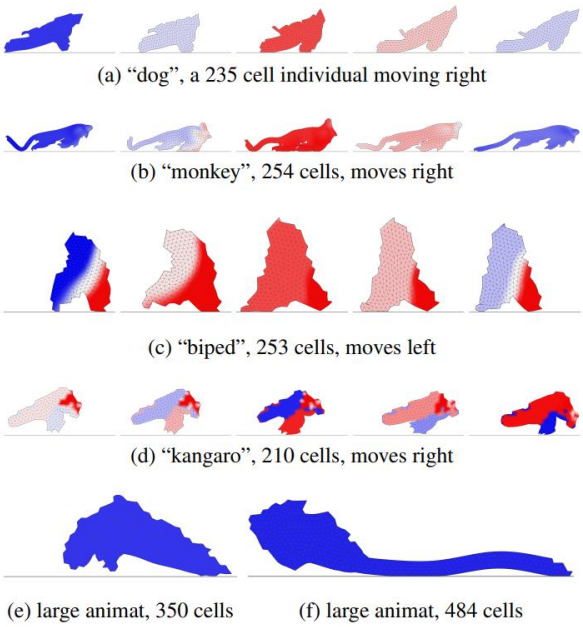


Fig.5. *Soft-bodied animats developed and controlled by the GRN (Joachimczak, Suzuki & Arita, 2014)*

1.2 Voxel-based Soft Robots

Voxel-based Soft Robots (VSRs) are modular robots consisting of voxels which can expand and contract. They have sparked interest amongst researchers, as they can exhibit animal-like behaviours, and their modular design could allow them to be more easily constructed in the physical world than other soft robots. (Hiller, Lipson, 2012)

1.2.1 Existing Simulators for Voxel-based Soft Robots

Open-source software has been developed to simulate VSRs, and they can provide a benchmark for comparing different models. The following tools have been used previously to simulate VSRs:

- **Voxelyze** (Hiller, Lipson, 2014) – This simulator is written in C++ and uses a mass-spring model. It can simulate 3D VSRs accurately with heterogeneous materials of differing stiffnesses and densities in a physically accurate environment. This software does not implement any control tasks.

  Voxelyze was used by (Cheney et al., 2014), where they evolved the robot's shape for locomotion using CPPN-NEAT, a generative encoding algorithm. Their robots consist of different materials with pre-specified frequencies to control actuation. They found that CPPN-NEAT can generate a diverse range of behaviours.

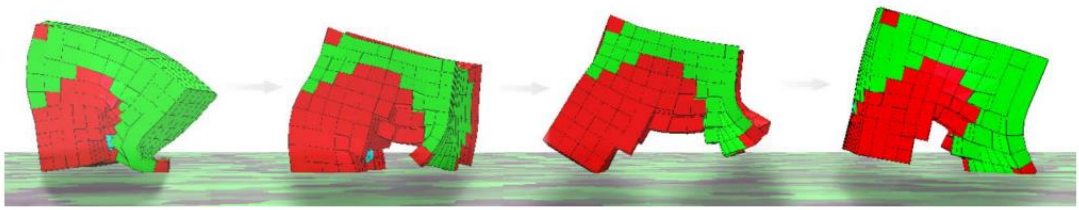


Fig.6. *A 3D VSR simulated in Voxelyze (Cheney et al., 2014)*

- **2D-VSR-Sim** (Medvet et al., 2020b) – This simulator is written in Java and uses a mass-spring model. It can simulate 2D VSRs and provides an implementation of a locomotion task. However, this simulator does not implement other tasks, such as object-manipulation.

  This simulator has been used to simulate VSRs on locomotion tasks. For example, (Pigozzi, Medvet, 2022) evolved an ANN neural controller and conducted experiments on two fixed robot shapes.

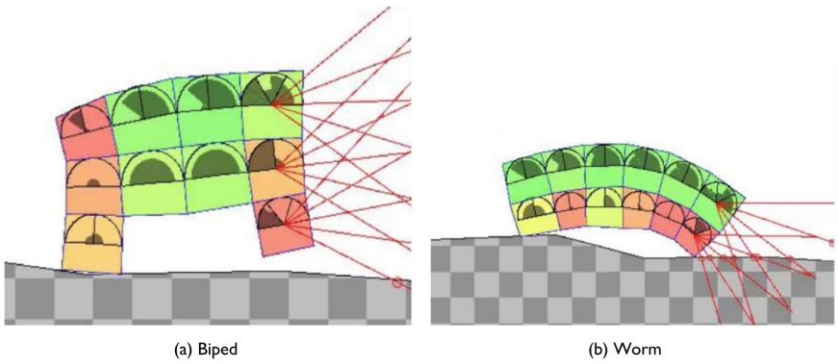


Fig. 7. *2D VSRs simulated using 2D-VSR-Sim (Pigozzi, Medvet, 2022)*

- **EvoGym** (Bhatia et al., 2021) – EvoGym was recently developed as a benchmark for comparing algorithms to co-design the body and brain of 2D VSRs. The software is written in C++ and uses a mass-spring model with an accessible Python interface. The robots can have different materials, such as rigid, soft, and actuator voxels. EvoGym implements more than thirty locomotion and object manipulation control tasks.

EvoGym also implements three benchmark co-design algorithms. In these algorithms, PPO optimises the controller for robot designs generated by a different design algorithm.

Fig.8. *2D VSR generated using EvoGym optimised for carrying (Bhatia et al., 2021)*

1.3 Key Takeaways

Studies have demonstrated that cell-based GRNs can control robots. However, these complex models have only been applied to simple rigid tasks such as a dot traversing a maze. (Asr, Majd, 2013)

In contrast, abstract GRNs, such as recurrent-based GRNs, have been used in more varying and challenging tasks, such as controlling soft-bodied animats on locomotion tasks. (Joachimczak, Suzuki & Arita, 2014) Therefore, it seems appropriate for this project to use an abstract model for controlling soft-bodied agents. While these abstract models do not implement the entire transcription and translation processes, they model the fundamental aspects: recurrent gene interactions.

Previous studies have used evolutionary algorithms to optimise the GRN, producing successful results on various problems. Therefore, this approach will be used in this project.

Considering different VSR simulators, EvoGym implements a variety of locomotion and object-manipulation tasks and includes benchmark co-design algorithms which are not present in other simulators. EvoGym is the most appropriate simulator for this

project and will be used to simulate the VSRs for the GRN and conventional PPO models.

1.4 Under-Researched Areas

Many studies that used GRNs were applied to rigid control tasks, such as controlling a fixed virtual car (Sanchez, Cussat-Blanc, 2014). However, biology has evolved soft-bodied animals made from flexible materials.

Most control tasks that are used with GRNs are purpose-built for each study. However, the lack of standardised tasks makes it difficult for researchers to compare the performance between different algorithms. A benchmark VSR simulator will be used to make a fair comparison between algorithms.

The previous studies on evolving soft robots only utilise locomotion-based tasks. What has not been explored is whether GRNs can be applied to control soft robots on other tasks, such as object manipulation.

Chapter 2: Replication of Previous Work

Before applying the GRN model to the EvoGym control tasks, the model and control tasks will be independently replicated alongside selected results from relevant papers.

2.1 Replicating A Suitable GRN Model

Having considered different GRN models in **Chapter 1**, the models that appear most feasible for evolving VSRs are abstract recurrent-based models. Despite not being as biologically realistic, they are inspired by the regulatory interactions that occur within cells. They have been shown to achieve good results on locomotion and obstacle-avoidance tasks. (Joachimczak, Suzuki & Arita, 2014) (Moreira, Rennó-Costa, 2021)

Generally, these approaches model the GRN as a directed graph of genes, where each gene activates or inhibits other genes through weighted edges. Each gene possesses an activity level representing how much of that gene exists. Genes are grouped into three classes: *sensors,* which take environmental inputs; *controllers,* which control the agent's actuators; and *processors,* which regulate gene activity. (Moreira, Rennó-Costa, 2021)

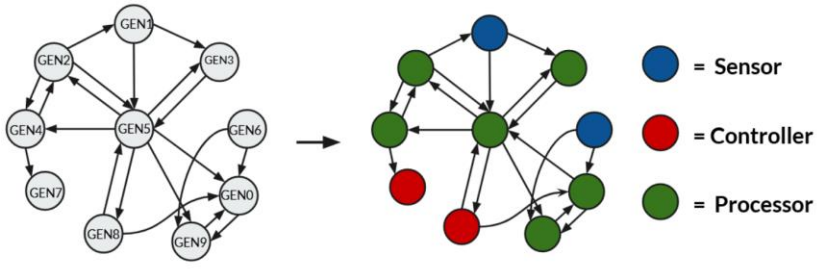


Fig.9. *The abstract GRN used by (Moreira, Rennó-Costa, 2021)*

One paper showed that recurrent-based GRNs can exhibit associative memory equivalent to a Hopfield network and can learn and recall multiple patterns. (Watson et al., 2014) Their model will be replicated and implemented in Python alongside some of their experiments.

2.1.1 Overview of Methodology

(Watson et al., 2014)'s Model

This paper models the individual's development using a recurrent GRN. The individual's phenotype is described by a set of N phenotypic traits at a given developmental time step P(t), represented by a vector of real numbers. The individual's genotype consists of two parts: a vector of direct effects on traits, G, and the elements b_{ij} of an interaction matrix, B. This can model an abstract GRN, where P represents a pattern of gene activity levels, and B represents a network of up and down regulatory interactions.

For every developmental step, the activity level of each gene p_i within the phenotype vector is updated according to the following equation:

$$p_i(t+1) = p_i(t) + \tau_1 \sigma\left(\sum_{j=0}^{n} b_{ij} p_j(t)\right) - \tau_2 p_i(t)$$

They chose τ_1 = 1 for the rate constant, which controls the magnitude of the interaction terms and τ_2 = 0.2 for the decay rate. They applied a non-linear function to the sum of weighted incoming connections using $\sigma(x) = \tanh(x)$.

The individual is developed for a set number of developmental time steps.

Evolutionary Model

Unlike conventional evolutionary algorithms, which evaluate a population of genotypes over several generations, (Watson et al., 2014) used a Hill-Climbing selection model. This is a simple optimisation technique which iteratively explores the local search space to find the optimal solution by making small mutations to the genome. In this technique, only mutations that result in better fitness from the previous iteration are accepted.

Experiments

The paper conducted experiments on the GRN to test its ability to learn bit-patterns:

- In Experiment 1, the GRN is tasked with learning a single pattern: [1,1,-1,-1,-1,1,-1,1].
- In Experiment 2, the GRN is tasked with learning two patterns: [1,1,-1,-1,-1,1,-1,1] and [1,-1,1,-1,1,-1,-1,-1]. During simulation, the current target pattern alternates between these arrays every 2000 iterations.

To evaluate the GRN's fitness, the phenotype is compared to the current target pattern; this is calculated by computing the dot product between the phenotype and the target.

Replicating the Model and Experiments in Python

Python has been chosen as the programming language to implement the GRN model since it is a high-level language with many libraries that can help build the solution. Two libraries will be used:

- NumPy will be used since it can generate random uniform values for mutating the genome. This library also contains special arrays optimised to work faster than traditional Python arrays; these will be used to store the genome.

- Matplotlib will also be used for its plotting features to visualise the data from the experiments and test that the model works correctly.

An iterative development and testing approach has been chosen to replicate the GRN model so that bugs can be identified and corrected quickly. This will be achieved with PyCharm IDE, which was chosen due to its built-in debugger with breakpoints. It is also easy to view the console output alongside the program in this IDE, which will be helpful for testing.

Unit Tests will be performed to test each method. Following this, a fitness graph will be generated to check that it increases monotonically, which is the expected result for Hill-Climbing. After testing, graphs will be plotted and compared against the results presented by the paper to verify that they match.

2.1.2 Implementation

Different functions were created to implement the GRN, such as one for calculating fitness and one for developing the genotype. This makes the code clearer and easier to read:

```python
def calculate_fitness(vec1, vec2):
    return np.dot(vec1, vec2)

2 usages
def develop(gene_potentials, interaction_matrix, T):
    gene_count = len(gene_potentials)
    phenotype = np.zeros(gene_count)

    for t in range(T):
        # Calculate new gene potentials
        m = 0
        for i in range(gene_count):
            sum_of_activities = 0
            for j in range(gene_count):
                sum_of_activities += (interaction_matrix[m] * gene_potentials[j])
                m += 1

            # Applies the GRN equation
            phenotype[i] = gene_potentials[i] + rate*math.tanh(sum_of_activities) - degradation_rate*gene_potentials[i]

    return phenotype
```

Fig.10. *Screenshot showing the develop() and calculate_fitness() functions*

After implementing each function, Unit Tests were conducted to verify their correctness: **See Appendix D.1 for Unit Tests**

The whole solution was tested by running an experiment and plotting the fitness graph. This initially produced a strange result where the fitness was unstable:

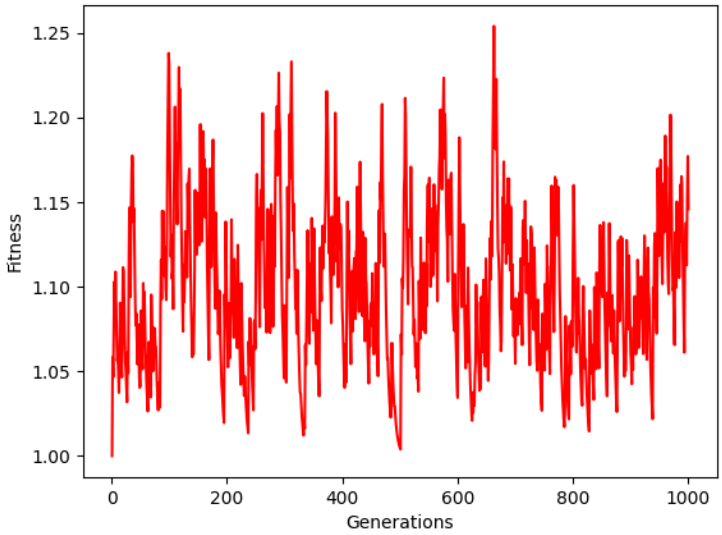


Fig.11. *Initial plot of the GRN's fitness*

This result suggests that there is a bug in the code since the fitness should increase monotonically. To investigate this, PyCharm's debugger was used to inspect the program.

During debugging, it was found that the result was stored back in the same array when the gene potentials were developed by the develop() function. This meant that when the mutated genotype was developed, the mutation was adopted before evaluating whether it was beneficial. This error was corrected by adding a separate phenotype array to store the result:

```
# Develop genotype
gene_potentials = develop(gene_potentials, interaction_matrix, max_time_step)
```

```
# Develop genotype
phenotype = develop(gene_potentials, interaction_matrix, max_time_step)
```

Fig.12. *The code before (top) and after (bottom) correcting the bug*

The test was repeated to check that the issue had disappeared. The results show the intended behaviour for Hill Climbing:

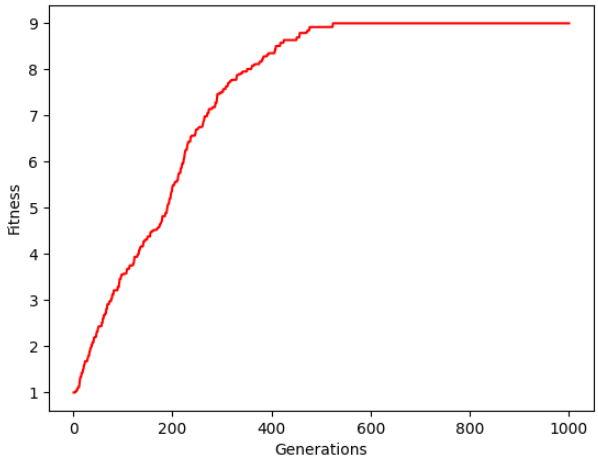


Fig.13. *Plot of the GRN's fitness after correcting the bug*

2.1.3 Results

After testing the code, Experiments 1 and 2 were replicated from (Watson et al., 2014), and the results were plotted to check that they match the paper's results:

Experiment 1: Single Selective Environment

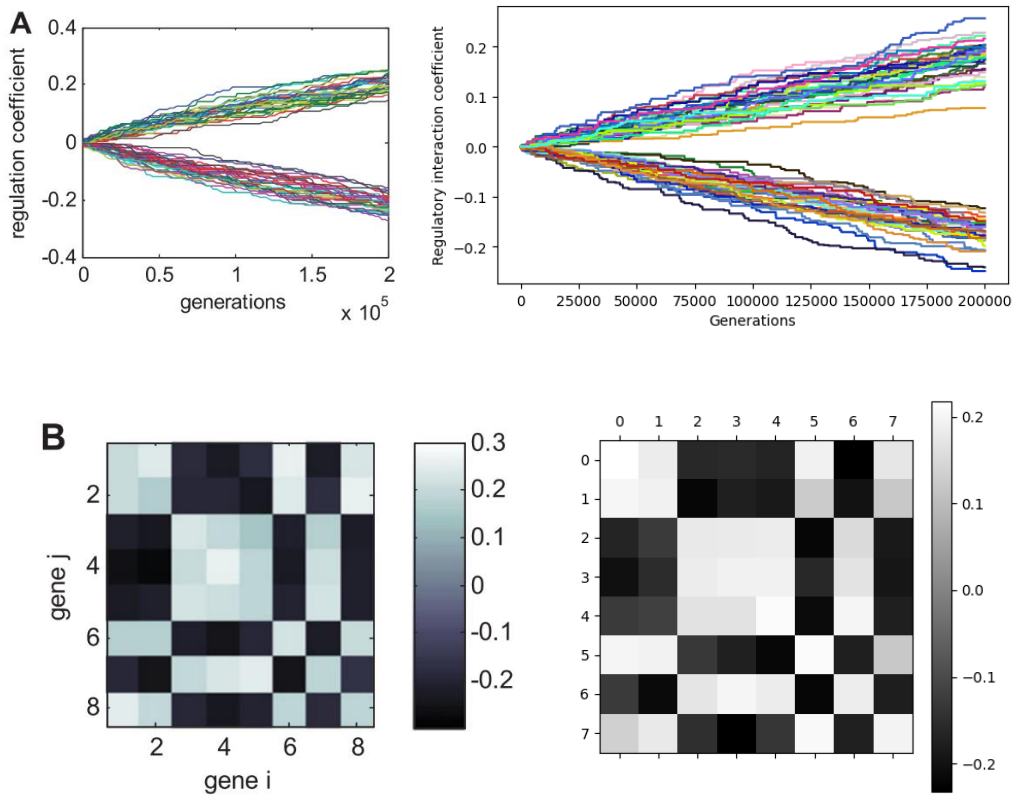


Fig.14. *Experiment 1: Graph of GRN's gene potentials and interaction matrix on the left (Watson et al., 2014) compared with my replicated graphs on the right*

Experiment 2: Varying Selective Environment: Multiple Memories

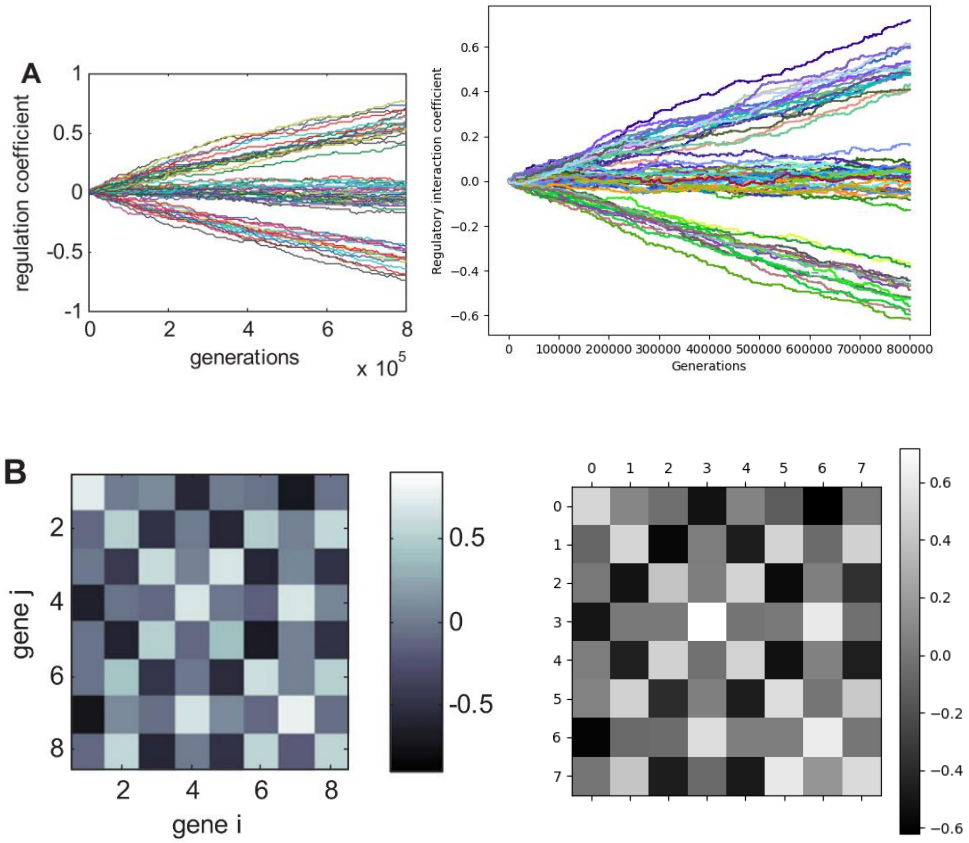


Fig.15. *Experiment 2: Graph of GRN's gene potentials and interaction matrix on the left (Watson et al., 2014) compared with my replicated graphs on the right*

2.1.4 Summary

Watson's GRN model has been successfully replicated and implemented in Python. The results show their model can exhibit memory capabilities by learning bit patterns.

2.2 Replicating A Suitable Control Task

An existing simulator will be replicated to provide a fair environment for conducting experiments. EvoGym was chosen since it has different locomotion and object manipulation control tasks which are not present in other simulators.

2.2.1 Overview of Methodology

EvoGym

EvoGym is a multi-material VSR simulator, where each robot can consist of actuators (vertical and horizontal) which expand and contract, as well as soft and rigid voxels which do not produce actuation. EvoGym provides over 30 control tasks, such as carrying a block and walking up a hill. Each task has a reward function which evaluates the robot's performance.

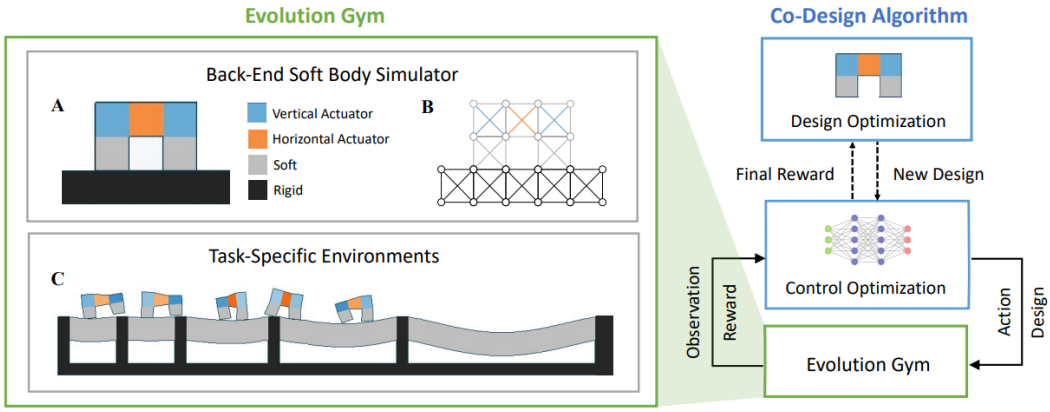


Fig.16. *Overview of EvoGym (Bhatia et al., 2021)*

Co-optimisation Algorithms

EvoGym provides three approaches for co-optimising the robot's shape and controller. PPO is used to optimise the controller in each approach and different design optimisation algorithms optimise the robot's shape: Bayesian Optimisation, Genetic Algorithm and CPPN-NEAT.

The paper found that PPO with Genetic Algorithm (PPO+GA) performed better than the other approaches. To verify the paper's results, PPO+GA will be replicated across a selection of control tasks.

2.2.2 Running EvoGym Experiments

Installation of EvoGym

EvoGym was installed from the GitHub repository by following the online instructions. A custom Python virtual environment was created to hold the simulator's dependencies.

Replicating PPO+GA with EvoGym

EvoGym provides over 30 different control tasks, and a selection of these will be replicated:

- Locomotion Tasks (Walker-v0, UpStepper-v0)
- Object Manipulation Task (Carrier-v0)

(Bhatia et al., 2021) conducted three tests for each environment using different seeds. Running the experiment multiple times provides a more accurate way of comparing algorithms; therefore, three experiments with different seeds will be replicated for each environment.

EvoGym contains files for running each benchmark algorithm, such as 'run-ga.py' for running the PPO+GA algorithm. Before running this file, the seed for the experiment was set so that independent samples could be collected.

The following command was executed to run each experiment, and the *-env-name* argument was modified to specify the control task. All of the remaining arguments were set the same as what was used in the paper to ensure a fair comparison:

```
python run_ga.py --env-name "Carrier-v0" --algo ppo --use-gae --lr 2.5e-4 --clip-param
0.1 --value-loss-coef 0.5 --num-processes 4 --num-steps 128 --num-mini-batch 4 --log-
interval 100 --use-linear-lr-decay --entropy-coef 0.01 --eval-interval 50
```

Fig.17. *Command that was executed to run PPO+GA on the 'Carrier-v0' task*

Initially, co-design experiments were replicated on my laptop with a CPU speed of 2.40GHz and 8 processors. With this setup, each generation of one experiment took ~1-2 hours, even when taking advantage of multiprocessing. Since some experiments needed to be run for 30 generations and replicated over 3 different seeds, it took over a week for the algorithm to be replicated on a single environment. Due to the high computational requirements, it was decided to reinstall EvoGym on a remote desktop so experiments could run overnight.

2.2.3 Results

The replication results have been plotted, indicating that the reward values achieved roughly match the values presented in the original paper. It is expected that the results are not exactly the same as the paper's results since these replication experiments were conducted using different seeds from what was used in the paper:

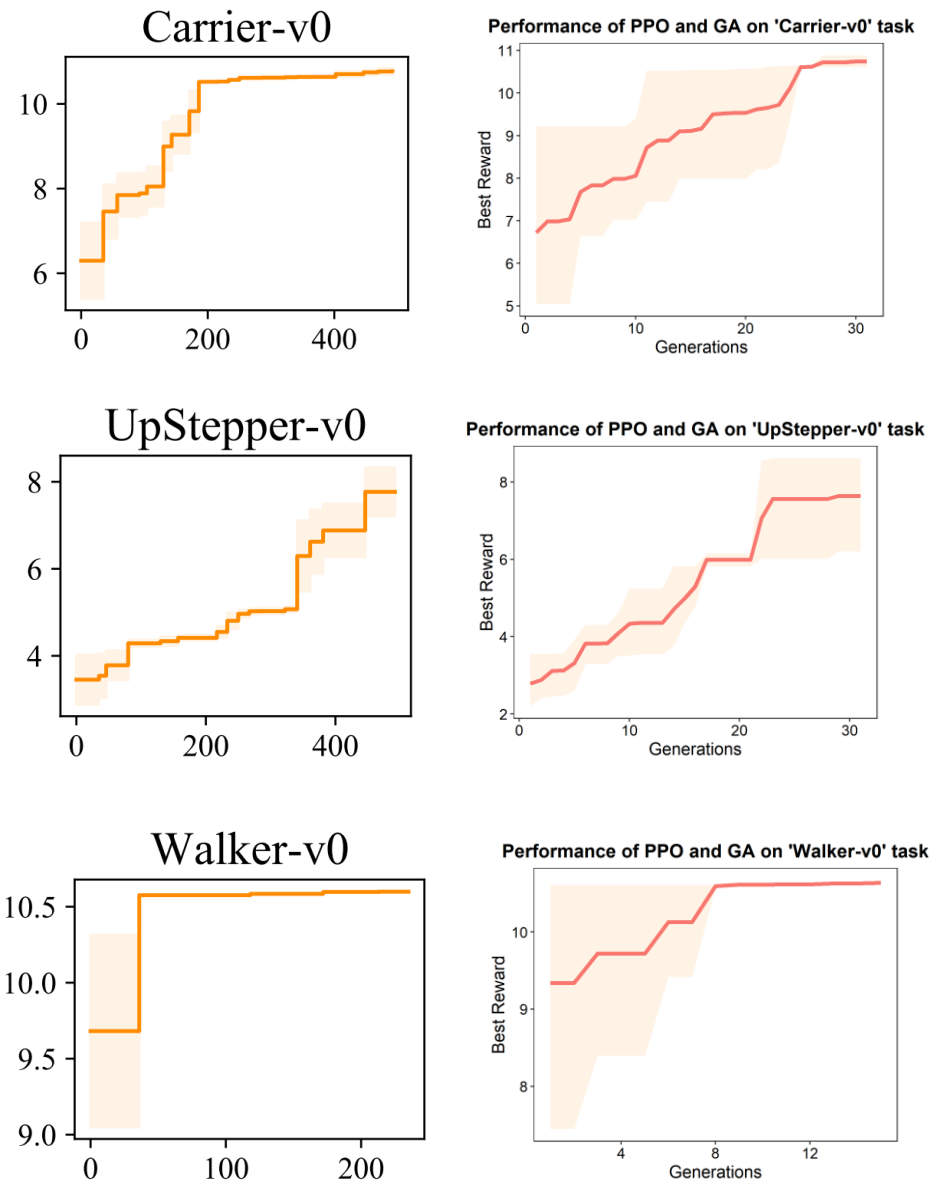


Fig.18. *Results taken from (Bhatia et al., 2021) are shown (left) compared with my replicated results (right). The solid line represents the average performance between 3 runs, and the shaded region represents the variance*

To verify that the algorithm produces valid robots, EvoGym's in-built visualiser was used to view the best-performing robot for each task:

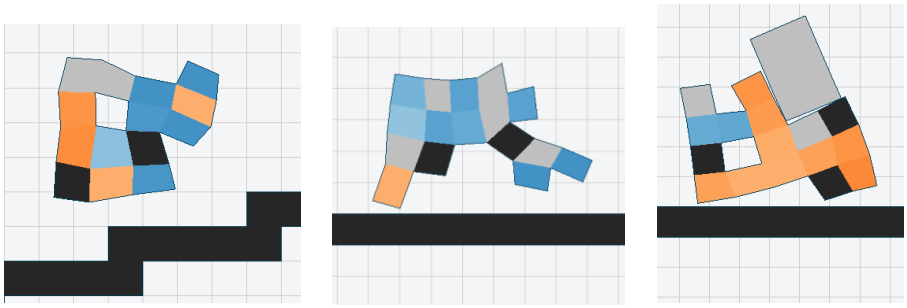


Fig.19. *The best-performing robot for each environment. UpStepper-v0 (left), Walker-v0 (centre) and Carrier-v0 (right).*

2.2.4 Summary

A selection of locomotion and object manipulation tasks from EvoGym have been replicated. The results demonstrate that the simulator and control tasks work correctly and can generate reproducible results.

Chapter 3: Application of the GRN Model

In this section, the GRN model will be implemented within EvoGym. This will involve designing, developing, testing, and integrating new classes to work with EvoGym so that GRN robots can be evolved and evaluated on EvoGym's control tasks.

Experiments will be conducted with the GRN and PPO controllers, and they will be applied to control different fixed robots for locomotion and object-manipulation tasks. Results will be compared and analysed to determine whether a GRN can effectively control a VSR and perform similarly or better than PPO.

3.1 Methodology

Task: Controlling Robots with Fixed Shapes

EvoGym provides two hand-designed robots:

- ***speed_bot*** – This robot has been designed for speed. It has 10 horizontal actuators, 5 rigid voxels and 1 soft voxel.
- ***carry_bot*** – This robot has been designed to carry an object. It has 8 horizontal actuators, 4 vertical actuators, 3 rigid voxels and 4 soft voxels.

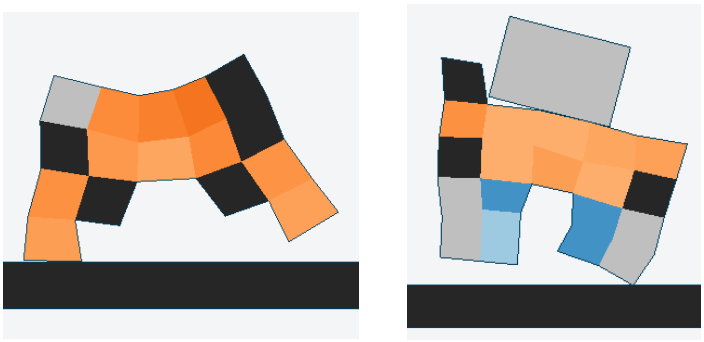


Fig.20. *speed_bot is shown (left) and carry_bot (right)*

These robots will be used to compare the performance of the GRN controller against the PPO controller. Results will be collected and analysed to evaluate the GRN's effectiveness.

The *speed_bot* will be evaluated on two locomotion tasks:

1. **Walker-v0:** This task evaluates the robot on a flat terrain with no obstacles. EvoGym classifies this as an easy task.
2. **UpStepper-v0:** This task evaluates the robot's ability to climb stairs of varying lengths. EvoGym classifies this as a medium task.

The *carry_bot* will be evaluated on one object manipulation task:

3. **Carrier-v0:** This task evaluates the robot's ability to catch and carry a box along a flat terrain. EvoGym classifies this as an easy task.

Genome Representation

To apply the GRN model to these robots, the number of genes within the GRN must be specified. In EvoGym, the observation space (i.e., the number of sensors provided

to the robot) is dependent on the shape of the robot and the control environment. The action space is equal to the number of actuators within the robot.

To ensure that the GRN has enough genes to map the observation and action spaces, the total number of genes will be calculated using this equation:

$$\text{size(observation\_space)} + \text{size(processor\_genes)} + \text{size(action\_space)}$$

observation_space and *action_space* can be determined by querying EvoGym's in-built methods. The number of processor genes is a model parameter that was set to 32.

The robot's genome consists of regulatory interactions between genes. This can be represented using an adjacency matrix of real values between -1 and 1, which will be flattened into a 1-dimensional array. With this configuration, a linear genome can represent a fully-connected recurrent network, where each gene has a regulatory effect on every other gene, including itself.

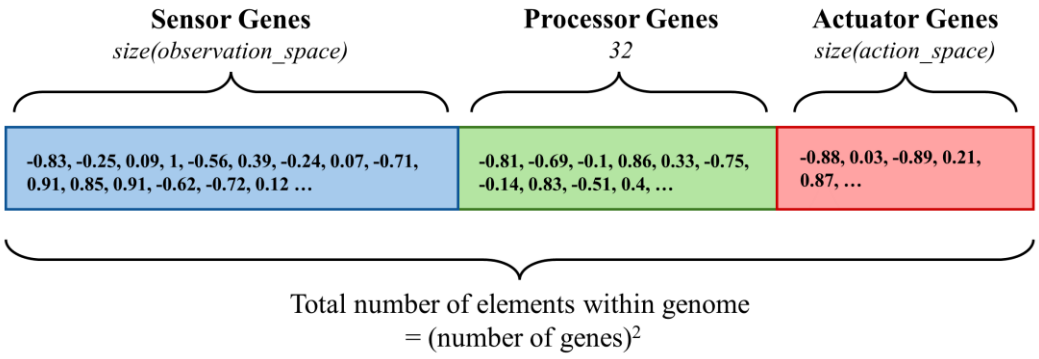


Fig.21. *Genome representation*

Evolutionary Algorithm

(Watson et al., 2014) used a Hill-Climbing selection model, which was shown to work well for learning simple bit-patterns. However, it is unlikely that Hill-Climbing would

find a suitable controller since it is ineffective at optimising problems with multiple local optima. (Prügel-Bennett, 2004)

Evolutionary algorithms can work on soft-bodied control tasks and will be used instead. This approach was successfully used by (Nadizar et al., 2023) to evolve different VSR controllers for locomotion tasks.

The following are the steps for the Evolutionary Strategy used by (Nadizar et al., 2023), which has been adapted to evolve a population of GRNs:

1. Create an initial population of $\mathbf{n_{pop}}$ individuals by generating a random interaction matrix for each individual, which consists of elements sampled from a uniform distribution between -1 and 1.
2. Evaluate each individual on an EvoGym task and retrieve their fitness.
3. Select the fittest ¼ of individuals as parents and compute the element-wise mean μ of their interaction matrices.
4. Copy the fittest individual to the new population for the next generation.
5. Generate offspring from μ by adding random Gaussian noise $N(0, \sigma)$ to each element. Repeat this until the new population size equals $\mathbf{n_{pop}}$
6. Repeat Steps 2 to 5 for each generation.

Fig.22. *Pseudocode for the Evolutionary Strategy*

The parameters for this algorithm are set to: $\mathbf{n_{pop}} = 36$ and $\boldsymbol{\sigma} = 0.35$.

Furthermore, the GRN's decay rate was reduced from 0.2 to 0.02 since this lower value was found to work better for EvoGym.

3.2 Implementation

Before writing any code, the problem was decomposed into smaller parts to determine the key requirements for the program. Decomposition makes it easier to understand what is needed to implement the solution.

3.2.1 Problem Decomposition

The following key components of the solution were identified:

Run Script	This will begin an evolutionary experiment and specify the parameters (such as population size, environment, and seed).
Evolutionary Algorithm	This will initialise a random population of robots and apply an Evolutionary Algorithm to the current population to determine which robots will be added to the next generation.
Robot	This class represents an individual robot.
GRN	This class represents the robot's controller. One instance of this class will be created for each robot.
Multiprocessing Manager	This will process fitness evaluations in parallel by taking advantage of multiprocessing, thereby reducing the experiment's runtime.

Fig.23. *Key components of the solution*

3.2.2 UML Diagrams

A UML Communication Diagram was created to visualise how the classes work together. The black boxes represent each component that will be implemented:

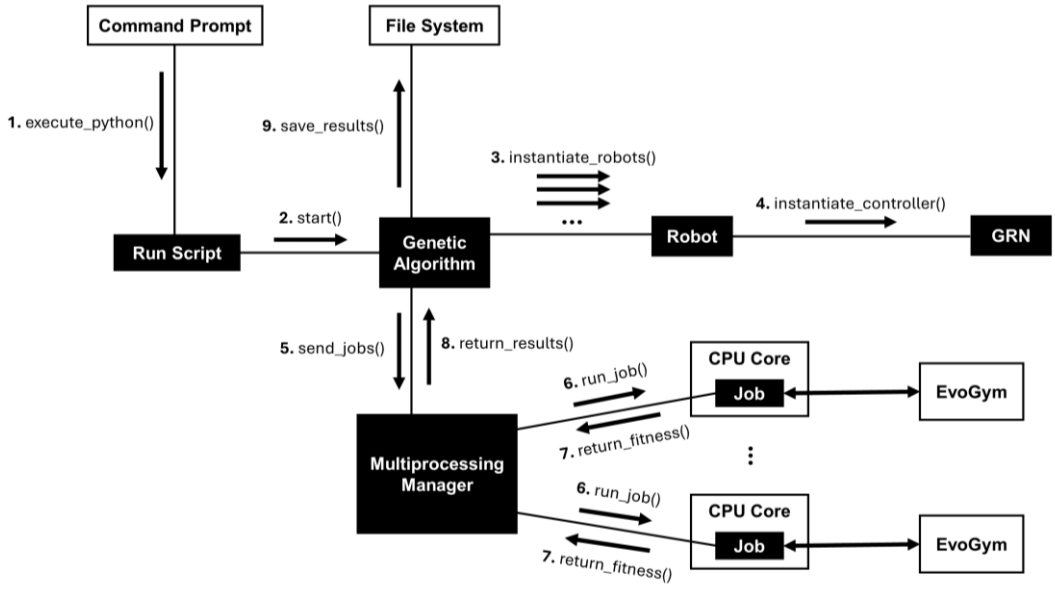


Fig.24. *UML Communication Diagram*

UML Class Diagrams were created to depict the required attributes and methods for the main classes:

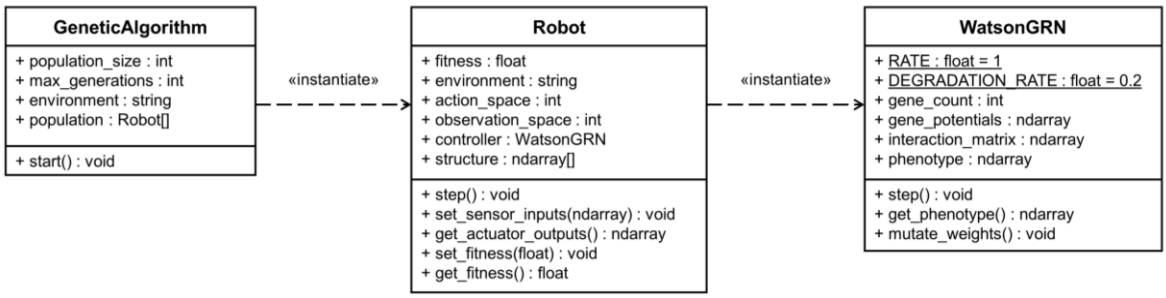


Fig.25. *UML Class Diagrams*

3.2.3 Programming and Testing

After designing the system, classes were implemented in Python via Object-Oriented Programming (OOP). Each class was placed into a different file to make it easier to manage the code.

Python was used since it is the language used by EvoGym, and using the same language allows the model to be easily integrated with their control tasks. OOP was chosen since it provides a modular framework, making adding or modifying functionality for individual classes easier without needing to rewrite the entire program. (Sushant Gaurav, 2023)

Unit Testing was applied by testing small code segments to verify their correctness, such as checking the step() method within the WatsonGRN class. **See Appendix D.2 for Unit Tests**

Integration Testing was then applied to check that modules work together correctly, such as testing the Robot and WatsonGRN classes simultaneously. Finally, System Testing was conducted to test the entire solution by observing the fitness values outputted.

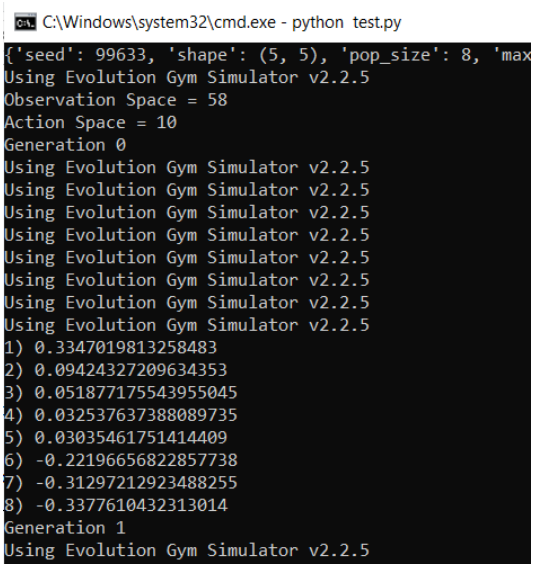


Fig.26. *Example results produced from System Testing*

After implementing the program, the Task Manager was inspected to verify that the program was applying multiprocessing correctly:

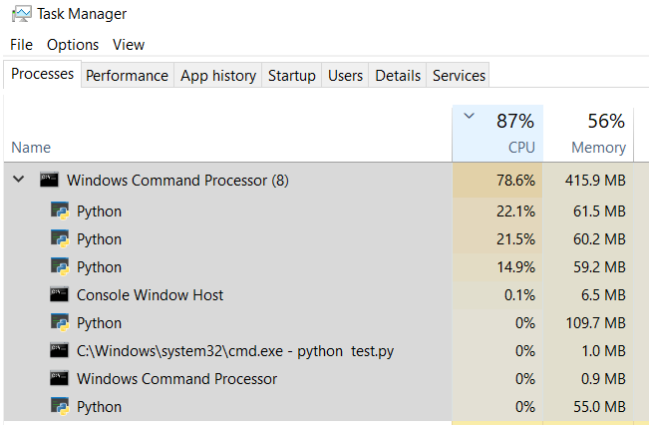


Fig.27. *Task Manager shows 2-3 active Python processes*

The Task Manager only showed 2-3 active Python processes. This suggests that the jobs were executed in series, meaning that multiprocessing was not working as intended.

After inspecting the code, an issue was identified that caused the GRN to be instantiated before jobs were sent to the Multiprocessing Manager. This used a lot of CPU time and increased the experiment's runtime.

The program was modified to eliminate this issue. The code responsible for instantiating the robot's controller in the Robot class was moved into a new class, which is only called to instantiate the robot's GRN during a process job rather than in the main program. After making this change, the test was repeated, and the Task Manager now shows multiprocessing working:

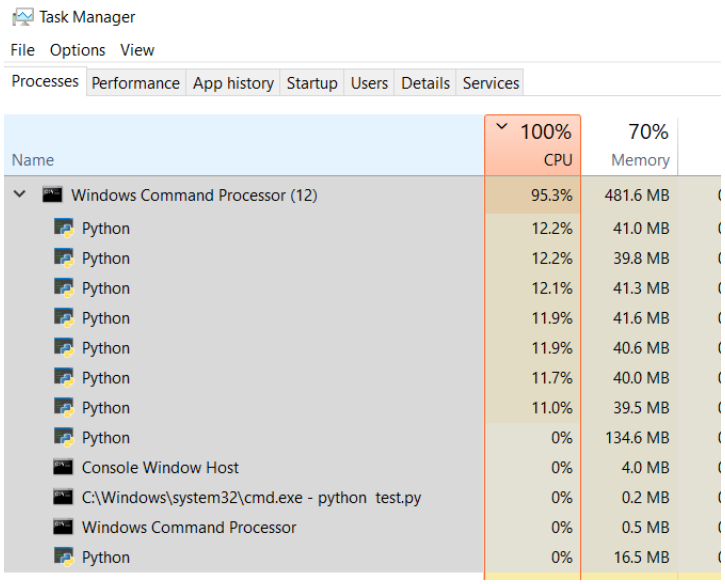


Fig.28. *Task Manager shows the modified program executing on many processes*

3.3 Experiments

To explore the behaviour and effectiveness of the implemented GRN controller, experiments were conducted on three control tasks: Walker-v0, UpStepper-v0 and Carrier-v0. These experiments were also conducted with a PPO controller, which allows the GRN to be compared against a conventional approach.

Each control strategy (PPO and evolved GRN) was replicated 10 times with different seeds to get a range of independent samples. In total, 60 experiments were conducted (30 for each control strategy).

3.3.1 Experiments with the PPO Controller

To set up the PPO experiments, the run_group_ppo.py file was edited to specify the parameters, including the seed, environment, and robot type. After this, the following command was executed with the default PPO parameters as used by (Bhatia et al., 2021):

```
python run_group_ppo.py --algo ppo --use-gae --lr 2.5e-4 --clip-param 0.1 --value-loss-coef 0.5 --num-processes 4 --num-steps 128 --num-mini-batch 4 --log-interval 100 --use-linear-lr-decay --entropy-coef 0.01 --eval-interval 50
```

Fig.29. *Command for running PPO*

3.3.2 Experiments with the GRN Controller

To set up the GRN experiments, the run script was modified to specify the experiment parameters (e.g. seed, environment, robot type and population size). After this, the program was executed to evolve a population of GRN robots.

3.3.3 Preliminary Results for GRN

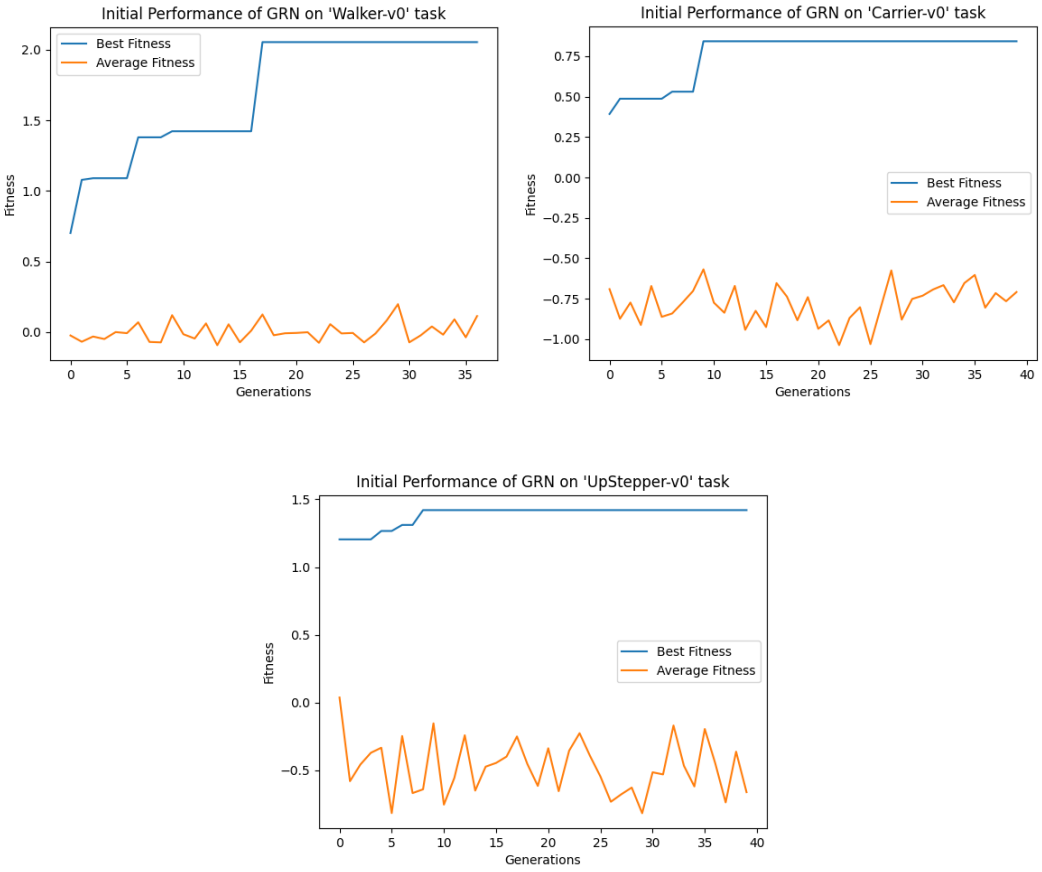


Fig.30. *Initial performance of GRN*

The GRN's initial best and average performance from a single experiment was plotted. These graphs show that the GRN's best fitness plateaus in every task, and the average fitness does not increase much throughout the experiment.

3.3.4 All Results

Performance of the PPO Controller

Results for **PPO** on Fixed Robot Shapes

Produced from 10 independent samples for each task

Task	Robot	Median	Best	Worst	IQR[1]
Walker-v0	speed_bot	10.5719	10.5789	10.5635	0.0039
UpStepper-v0	speed_bot	1.4416	1.6187	1.4040	0.0592
Carrier-v0	carry_bot	6.0312	10.5357	3.5226	2.3822

[1] IQR = Interquartile range

Fig.31. *Performance of PPO*

These results indicate that PPO can effectively optimise the 'speed_bot' for the Walker-v0 task, achieving a median fitness of 10.57. PPO has also shown good performance when applied to the 'carry_bot' for the Carrier-v0 task, achieving a median fitness of 6.03. However, it performed poorly when optimising the 'speed_bot' for the UpStepper-v0 task and only achieved a median fitness of 1.44.

Compared with the replication results conducted in **Chapter 2**, PPO generally does not perform as well at controlling fixed shapes compared to when coupled with a design optimisation algorithm. The only exception is the Walker-v0 task, where PPO achieved the maximum fitness possible by reaching the end of the environment.

Initial Performance of the GRN Controller

Initial Results for **GRN** on Fixed Robot Shapes

Produced from 10 independent samples for each task

Task	Robot	Median	Best	Worst	IQR[1]
Walker-v0	speed_bot	1.9014	3.3166	1.4347	0.3364
UpStepper-v0	speed_bot	1.4076	1.4237	1.3527	0.0260
Carrier-v0	carry_bot	0.8985	1.3042	0.5260	0.2777

[1] IQR = Interquartile range

Fig.32. *Initial results for the GRN*

These results show that the GRN achieved much lower fitness scores than PPO in every task. **See Appendix E.1 for full performance graphs**

3.3.5 Analysis

To investigate the GRN further, the robot's behaviour during simulation was analysed. Following this, a statistical test was conducted to show whether there is a statistically significant difference between the PPO and GRN results.

Behaviour and Network Analysis

A video of the best-performing robot's behaviour was inspected. This was taken from the Carrier-v0 experiment:

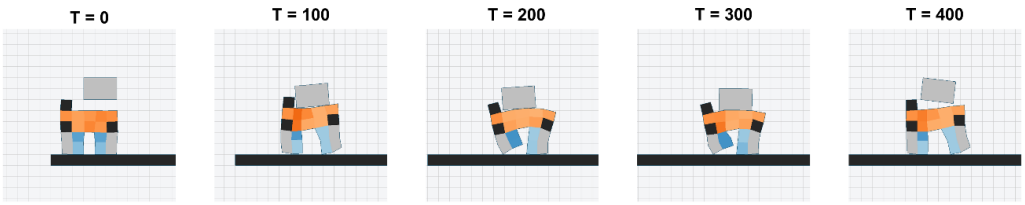


Fig.33. *GRN's performance on 'Carrier-v0'*

The video reveals that the GRN robot can move the block without dropping it; however, it moves quite chaotically and slowly.

A graph was created to plot the robot's expression levels for each actuator (each colour represents one actuator). This shows that the actuator expression levels tend to fluctuate unpredictably:

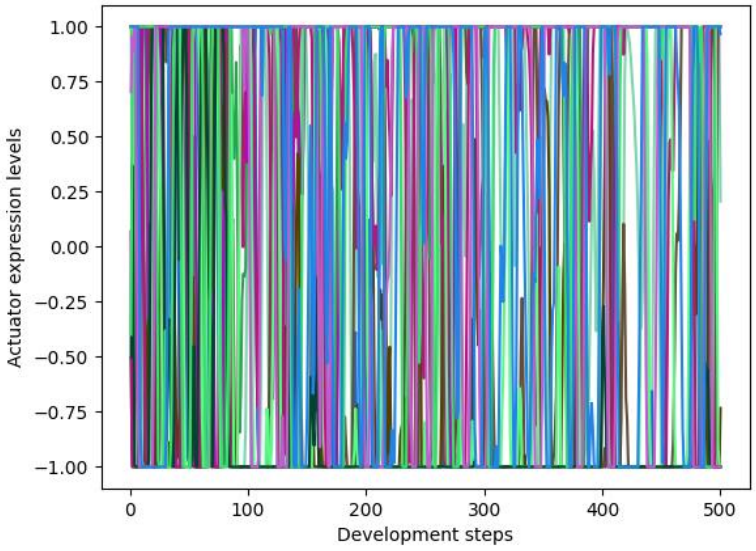


Fig.34. *GRN's actuator levels for 'Carrier-v0'*

The best-performing robot for the UpStepper-v0 and Walker-v0 tasks shows similar behaviour and noisy actuator levels.

See Appendix F.1 for further behaviour and network analysis

Statistical Analysis

To make a concrete comparison between the GRN and PPO controllers, a statistical test was performed for each control task. The Mann-Whitney U Test is a non-parametric statistical test which tests for differences between two independent groups (McKnight, Najab, 2010). This test can determine whether there is a statistically significant difference between the results of each control strategy.

For this test, the Null Hypothesis = There is no difference in the median fitness between the GRN and PPO. This test was conducted in R using the built-in wilcox.test() method:

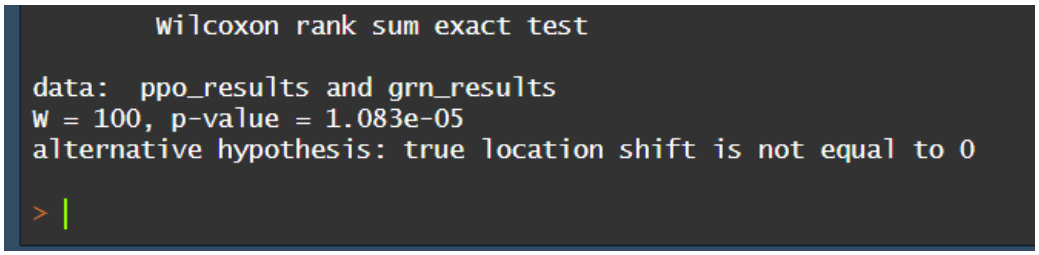


Fig.35. *Example R output for the Mann-Whitney U Test*

This test produced p-values of 0.00001083 for the Walker-v0 and Carrier-v0 tasks and a p-value of 0.0004871 for the UpStepper-v0 task.

These p-values are less than the standard 0.05 significance threshold; therefore, the Null Hypothesis can be rejected in favour of the Alternative Hypothesis which is that there is a statistically significant difference between the performance of PPO and GRN. Since the GRN's median performance is less than the PPO, this test shows statistically that the GRN performed worse than PPO.

See Appendix G.1 for box plot comparison charts

3.3.6 Conclusions

While the GRN has shown some effectiveness at controlling a VSR on locomotion and object-manipulation tasks, the results and analysis demonstrate that the GRN performs worse than PPO.

One reason could be that the evolutionary algorithm fell into a local optimum and could not find a better solution. The next chapter will consider different approaches to improve the GRN's performance.

Chapter 4: Refinement of the GRN Model

This section explores different methods to improve the GRN's performance. This was achieved through an iterative approach to update and test the program, perform experiments, collect results, perform analysis, and summarise the findings.

4.1 Refinement Iteration One

The results from **Chapter 3** show that the GRN model can control a fixed-shape VSR, but it performs significantly worse than PPO. The GRN's fitness plateaus, which suggests that the Evolutionary Algorithm (EA) may have fallen into a local optima. There are several reasons which may have caused this algorithm to perform poorly:

- Firstly, this algorithm initialises the entire genome with random values. This configuration means that the GRN is fully-connected, where every gene regulates every other gene. This probably caused the network to be overly saturated with noise, which prevented the robot from exhibiting effective actuation.

- Secondly, a high mutation value is applied with a small population size of 36. It is likely that this population size is too small to allow for solutions which can represent the different parts of the fitness landscape. (Aston et al., 2017)

A different EA will be considered to refine the GRN model.

4.1.1 Alternative Approach

To address the problem of the GRN being overly saturated, the new approach will only initialise 10% of the connections to be randomly connected. The mutation value **σ** will also be reduced for the same reason, and the population size will be increased to allow for more solutions.

The selection process will also be modified to increase population diversity via Tournament Selection. Tournament Selection works by choosing individuals from the population and selecting the best individual. (Razali, Geraghty, 2011)

Finally, a crossover operator will be implemented based on (Sims, 1994), who applied crossover to evolve rigid creatures. Introducing crossover should improve the convergence of the EA since individuals will inherit beneficial traits from their parents. (Hassanat, Alkafaween, 2017)

The following are the steps for the new EA:

1. Create an initial population of n_{pop} individuals by generating an empty interaction matrix for each individual. Set 10% of the matrix connections to a random value sampled from a uniform distribution between -1 and 1.
2. Evaluate each individual on an EvoGym task and retrieve their fitness.
3. Select the fittest ¼ of individuals as parents.
4. Copy the fittest individual to the new population for the next generation.
5. Generate offspring by either applying crossover and mutation to 2 randomly chosen parents OR apply mutation only to 1 randomly chosen parent (with equal likelihood).
6. For crossover, the crossover operator presented in (Simms, 1994) is applied to combine two parents.
7. For mutation, add Gaussian noise N(0, **σ**) to 5% of matrix connections, and set 1% of matrix connections to 0.
8. Repeat Steps 5 to 7 until the new population size equals n_{pop}
9. Repeat Steps 2 to 5 for each generation.

Fig.36. *Pseudocode for the refined EA*

For this algorithm, $\mathbf{n_{pop}} = 100$, $\boldsymbol{\sigma} = 0.1$ and Tournament Selection is applied with tournament size = 2 for Step 5.

4.1.2 Implementation

Implementing Crossover

(Sims, 1994) proposed a crossover operator which combines two directed graphs. For this operator, the nodes of both parents are aligned, and the parents' nodes and connections are copied to the child graph. Crossover points determine the point where the child switches from inheriting nodes from the first parent to the second:

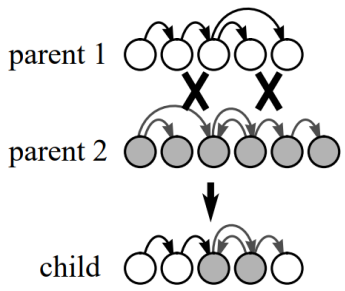


Fig.37. *Crossover operation (Sims, 1994)*

Their operator has been implemented by performing single-point crossover. This code was developed as a static method in the existing GRN class:

```python
# Crossover operator for GRN controller (Requires both GRNs to have the same gene count)
# Based on Karl Sims (1994) crossover operator for Mating Directed Graphs
def CrossoverGRN(parent_one, parent_two):
    if parent_two.gene_count != parent_one.gene_count:
        print('ERROR - PARENT GRNs DO NOT HAVE THE SAME LENGTHS!')
        return False

    child_gene_count = parent_one.gene_count
    child_interaction_matrix = np.zeros(child_gene_count * child_gene_count)

    # Seperate interaction matrix into gene connection arrays for each parent
    parent_one_genes = []
    parent_two_genes = []
    for i in range(0, parent_one.gene_count):
        parent_one_genes.append(np.take(parent_one.interaction_matrix, range(i*parent_one.gene_count,
                                                                             (i+1)*parent_one.gene_count)))
    for i in range(0, parent_two.gene_count):
        parent_two_genes.append(np.take(parent_two.interaction_matrix, range(i*parent_two.gene_count,
                                                                             (i+1)*parent_two.gene_count)))

    # Select random crossover point
    crossover_point = random.randint(0, child_gene_count)

    # Perform crossover
    for i in range(0, child_gene_count):
        if i < crossover_point:
            # Use genes from parent one
            gene_connections = parent_one_genes[i]
        else:
            # Use genes from parent two
            gene_connections = parent_two_genes[i]

        for j in range(0, child_gene_count):
            child_interaction_matrix[(i * child_gene_count) + j] = gene_connections[j]

    # Return new GRN with crossover
    child_grn = WatsonGRN(child_gene_count)
    child_grn.interaction_matrix = child_interaction_matrix.copy()
    return child_grn
```

Fig.38. *My implementation of (Sims, 1994)'s crossover operator*

Modifying the Evolutionary Algorithm

The EA class was modified to implement the new approach. Tournament selection was implemented using Python's random library to choose two randomly sampled individuals from the parents. Then the parent with the highest finesses is selected:

```python
# Selecting parent using tournament selection (size = 2)
tournament = random.sample(parents, 2)
parent = max(tournament, key=attrgetter('fitness'))
```

Fig.39. *Code for implementing Tournament Selection*

4.1.3 Testing

To verify that my crossover implementation works, two GRNs were generated with dummy data, and crossover was applied to them multiple times. Matplotlib was used to visualise the results of each child GRN:

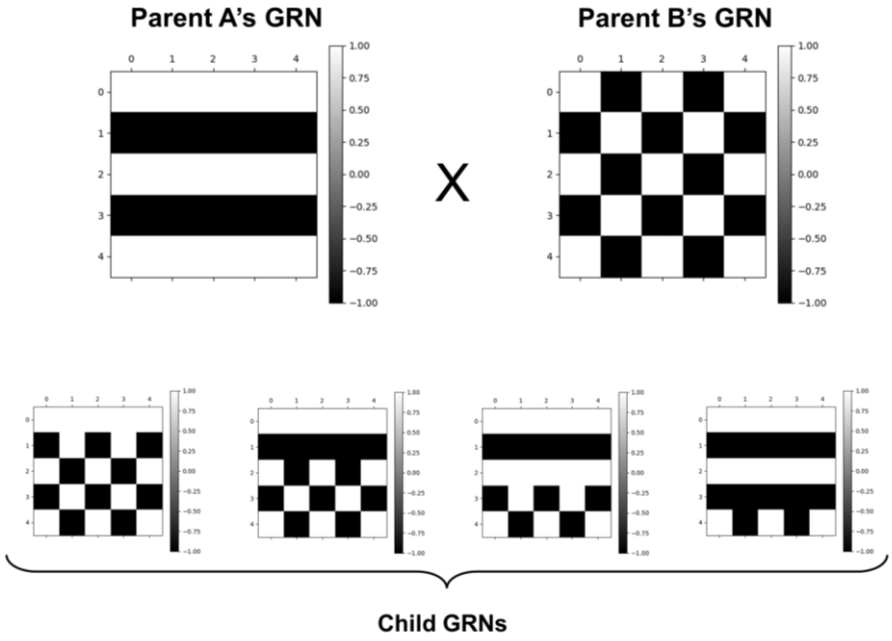


Fig.40. *Results from the crossover test*

This shows that the implemented crossover operator correctly combines elements from both parents and selects a random crossover point.

Further Unit Tests were applied to test the newly-implemented methods: **See Appendix D.3**

4.1.4 Experiments

Experiments were conducted on the refined GRN model to evaluate its effectiveness, and the same tasks and robots were used as in **Section 3.3**.

4.1.5 Preliminary Results

The results of this approach have shown quite a dramatic improvement compared with the previous GRN:

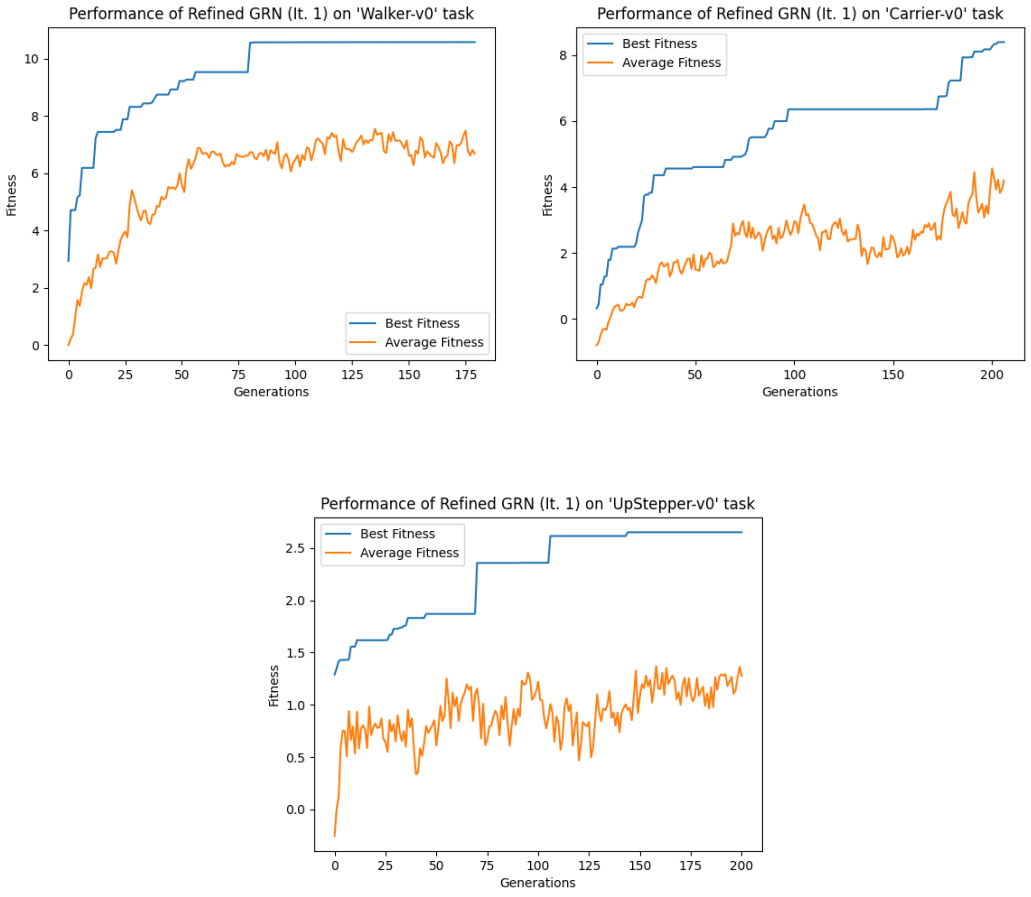


Fig.41. *Refined GRN's performance*

Unlike the previous iteration, where the fitness scores plateau after a few generations, the refined GRN's fitness does not plateau initially. However, the fitness did plateau in the 'Carrier-v0' experiment before continuing to increase again.

The 'Carrier-v0' and 'UpStepper-v0' experiments were run for 200 generations since the fitness score continued to increase. The reason for stopping the experiment at this time was due to the computational demands; one experiment took ~8 hrs to evaluate on task for 200 generations.

The ‘Walker-v0’ experiment was terminated after showing no further improvements after half of the simulation’s runtime.

4.1.6 All Results

Ten experiment samples of the refined GRN were conducted, and each experiment was run for a maximum of 200 generations or less if the fitness score did not improve in the last 50% of the runtime.

Results for **GRN** on Fixed Robot Shapes (Refinement Iteration 1)

Produced from 10 independent samples for each task

Task	Robot	Median	Best	Worst	IQR[1]
Walker-v0	speed_bot	10.5735	10.5775	9.0784	0.0100
UpStepper-v0	speed_bot	1.7140	2.6511	1.4744	0.3354
Carrier-v0	carry_bot	6.8021	8.2526	5.2195	2.0530

[1] IQR = Interquartile range

Fig.42. *Refined GRN results*

These results show that the GRN achieved a median performance comparable to that of the PPO on the Walker-v0 task. For the UpStepper-v0 task, the GRN achieved a greater median and best fitness performance than the PPO. For the Carrier-v0 task, the GRN achieved a lower best fitness performance than the PPO but obtained a higher median fitness. **See Appendix E.2 for full performance graphs**

4.1.7 Analysis

Behaviour and Network Analysis

A video of the best-performing robot for each task was inspected. This was taken from the Walker-v0 experiment:

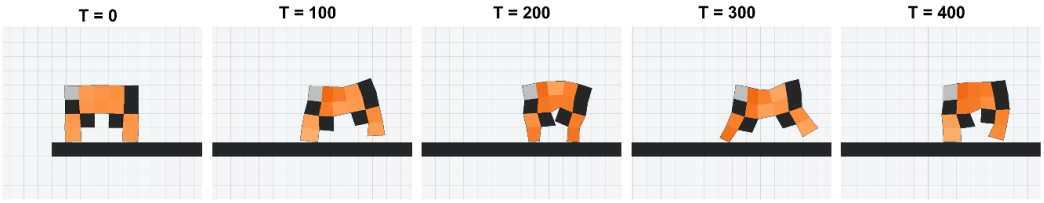


Fig.43. *Refined GRN's performance on 'Walker-v0'*

The video reveals that the robot runs quickly and successfully reaches the end of the environment. The actuators contract and expand repeatedly, which results in an effective running behaviour.

The actuator expression levels for this robot have been plotted, and this shows that certain actuators quickly fall into a regular cyclical pattern:

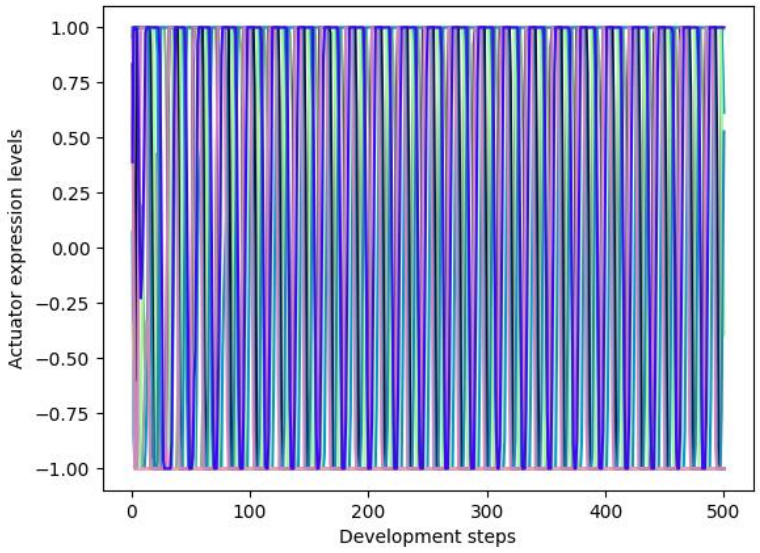


Fig.44. *Refined GRN actuator levels for 'Walker-v0'*

The best-performing robot for the UpStepper-v0 and Carrier-v0 tasks also shows effective actions.

See Appendix F.2 for further behaviour and network analysis

Statistical Analysis

A Mann-Whitney U test was performed to compare the refined GRN and PPO performance. This test produced p-values of 0.5787 for the Walker-v0 task, 0.6842 for the Carrier-v0 task, and 0.003886 for the UpStepper-v0 task.

The p-values for Walker-v0 and Carrier-v0 are greater than the standard 0.05 threshold; therefore, there is no statistically significant difference between the performance of the refined GRN and PPO for these tasks.

However, the p-value for UpStepper-v0 is less than 0.05; therefore, there is a statistically significant difference between the groups for this task. Since the GRN's median performance for this task is greater than that of PPO, this test shows statistically that the GRN performed better than PPO for the UpStepper-v0 task.

See Appendix G.2 for box plot comparison charts

4.1.8 Conclusions

The changes that were made to refine the GRN controller have resulted in robots which show effective control in locomotion and object-manipulation tasks. The robots performed better in every task than the previous iteration, and the statistical test showed that the GRN performed better than the PPO for the UpStepper-v0 task.

However, when evaluated on the UpStepper-v0 and Carrier-v0 tasks, the GRN achieved a lower fitness when compared to the PPO+GA co-design approach. This is also the case for the PPO controller (see **Chapter 3**). This suggests that the robot

designs may have prevented the GRN from achieving a higher fitness by constraining its behaviour to a fixed morphology.

A different approach will be considered in the next iteration to co-evolve the robot's controller and shape using a GRN.

4.2 Refinement Iteration Two

While the previous GRN effectively controlled the robot's movements, it performed worse than the PPO+GA co-design approach. This suggests that the fixed hand-designed robots are not perfectly suited for EvoGym's tasks.

One solution could be co-optimising the robot's controller and design using a GRN; this approach may be able to find better robot shapes.

4.2.1 Alternative Approach

In biology, GRNs not only control cellular behaviour but they also determine how cells develop to form complex multicellular animals. (Peter, Davidson, 2011). This idea has been extended to virtual animats by (Joachimczak, Suzuki & Arita, 2014), who showed that multicellular soft-bodied animats can be developed from a single cell using a GRN and controlled on locomotion tasks.

In this section, a GRN developmental process based on this previous work will be created and adapted for EvoGym. My strategy extends their work by evaluating robots on locomotion and object manipulation tasks and utilising EvoGym's

simulator. My strategy will also implement a decentralised controller adapted from (Medvet et al., 2020a) to allow voxels to communicate.

GRN Developmental Approach

Each robot starts as a single voxel in the centre of a 5x5 grid. This voxel contains a GRN made of 4 input and output genes and 32 processor genes.

During the developmental process, some processor genes have special functions:

- One gene instructs the voxel to divide when its activity exceeds a pre-specified threshold.
- Two other genes indicate the direction in which the divided cell should be placed relative to the voxel (left, right, up, or down).
- Four genes determine the voxel's fate, each representing the propensity for the voxel to become each of the four possible material states. When the voxel matures, its fate is determined by the gene with the highest activity.

The remaining processor genes serve to regulate the activity of other genes.

When a voxel divides, its GRN controller is copied into the newly-created voxel. Additionally, voxels can communicate with their neighbours through signalling input and output genes positioned on each side of the voxel.

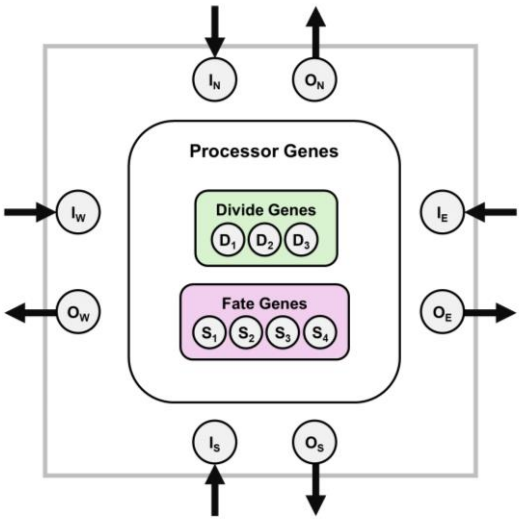


Fig.45. *The decentralised GRN controller*

This developmental process is simulated for 100 steps. Once the robot has fully developed, it is placed in EvoGym's environment, and every GRN in each of the robot's voxels are stepped. During simulation, voxels retain their signalling genes to allow for intercellular communication, and actuators have a single gene to control actuation.

4.2.2 Implementation

The current program will be amended to implement the required features for this new model.

In this approach, multiple voxels operate independently within the robot both during and after the developmental process. To model this, a new class: RobotVoxel will be developed, which allows one Robot to have many instances of RobotVoxel.

This class will have attributes such as its GRN controller and a location, which will be represented as a (row, column) tuple to store the voxel's location within the robot. This will allow voxels to send signals to their neighbours.

Additionally, a new attribute will be added to the Robot class for storing RobotVoxel instances so that the robot's voxels can be simulated. This class will also initialise one RobotVoxel at the centre of the grid to start the developmental process.

When generating the robot's shape, some robots may have developed no actuators, meaning they are unviable for EvoGym. To address this, these robots will receive an immediate penalty fitness of -100.

4.2.3 Testing

Before conducting experiments, Unit Tests were applied to test the newly-implemented methods: **See Appendix D.4**

The program was then tested to verify that it works as intended by developing different robot designs. This was achieved by creating many random robots and visualising what is produced:

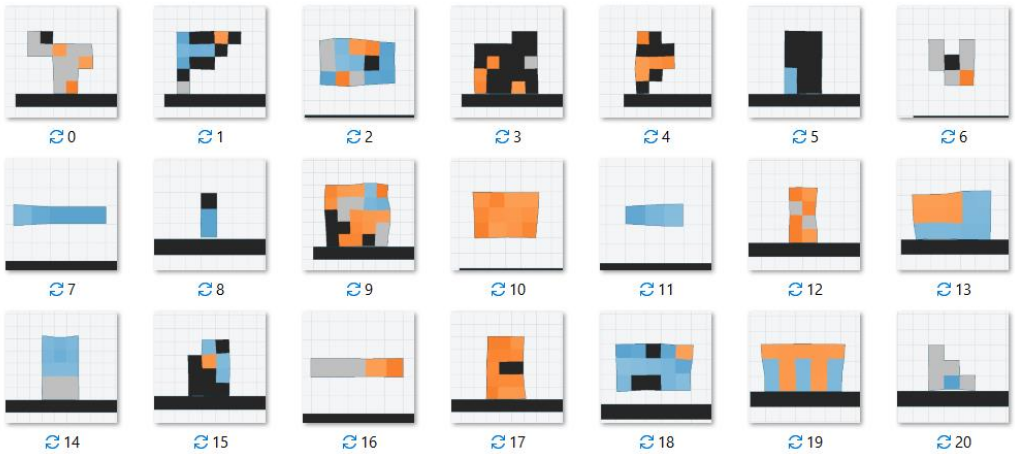


Fig.46. *Results from testing my developmental process*

The results show that the program successfully creates various robot designs from random genomes.

4.2.4 Experiments

Previously, the GRN was compared against PPO; however, since both the robot's controller and design will be evolved, it will be compared against the PPO+GA co-design algorithm, which was replicated in **Chapter 2**. This approach ensures that the co-design GRN can be fairly compared against a baseline co-design approach.

The evolutionary algorithm used previously was applied; however, the population size was increased to 200. This is because optimising the controller and design together is a more challenging problem than optimising the controller alone, so the higher population size will allow more solutions to be represented.

Experiments were conducted on the 'Walker-v0' and 'Carrier-v0' tasks.

4.2.5 Preliminary Results

The preliminary results show that the co-evolved GRN's best fitness increases in both tasks. The average fitness initially increases sharply as the unviable robots which produce no actuators are removed; following this, the average fitness appears to stagnate:

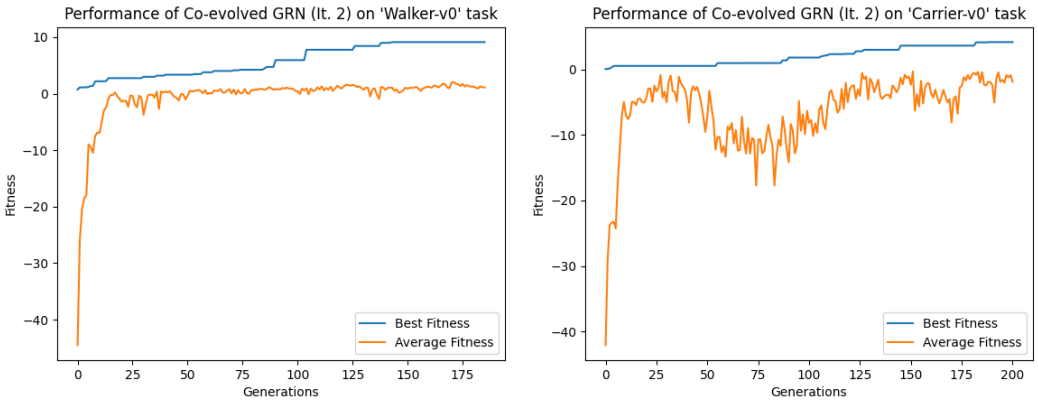


Fig.47. *Co-evolved GRN's performance*

The average fitness appears to skew the results for the best fitness; therefore, the graphs were re-plotted without the average fitness. This shows that the best fitness increases:

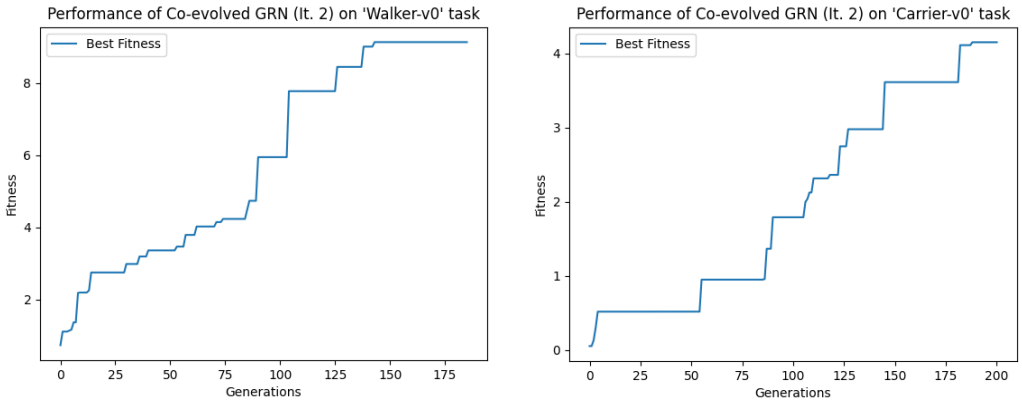


Fig.48. *Co-evolved GRN's performance (without average)*

4.2.6 All Results

Performance of the conventional PPO+GA approach

Three samples of the conventional PPO+GA approach were collected by replicating EvoGym's results (see **Section 2.2.3**).

Results for Co-optimised **PPO+GA**				
Produced from 3 independent samples for each task				
Task	Median	Best	Worst	IQR[1]
Walker-v0	10.6226	10.6676	10.6100	0.0288
Carrier-v0	10.7616	10.8815	10.5986	0.1415
[1] IQR = Interquartile range				

Fig.49. *PPO+GA results*

These results show that for both tasks, the PPO+GA co-optimisation approach performs better than when PPO is applied to control fixed robots.

Performance of the co-evolved GRN

Ten samples of the co-evolved GRN were conducted, and each experiment was run for a maximum of 200 generations or less if the fitness score did not improve in the last 50% of the runtime.

Results for Co-evolved **GRN** (Refinement Iteration 2)				
Produced from 10 independent samples for each task				
Task	Median	Best	Worst	IQR[1]
Walker-v0	6.6351	10.5933	4.2408	3.2124
Carrier-v0	1.4555	4.1512	0.2964	0.8883
[1] IQR = Interquartile range				

Fig.50. *Co-evolved GRN results*

These results show that the co-evolved GRN achieved a best fitness of 10.59 for the Walker-v0 task, which is comparable to PPO+GA; however, the GRN achieved a lower median score. For the Carrier-v0 task, the GRN achieved a lower performance than PPO+GA. **See Appendix E.3 for full performance graphs**

4.2.7 Analysis

Behaviour and Network Analysis

A video of the best-performing robot for each task was inspected. This was taken from the Walker-v0 experiment:

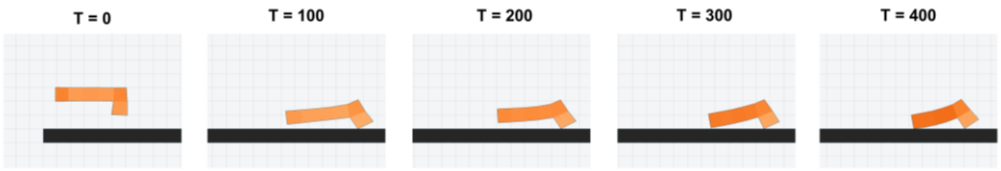


Fig.51. *Co-evolved GRN's performance on 'Walker-v0'*

Despite only having 6 voxels, the robot performs well: it repeatedly expands and contracts its central actuators, which allows it to move quickly and complete the task.

This cyclical behaviour is also observed in its actuator levels:

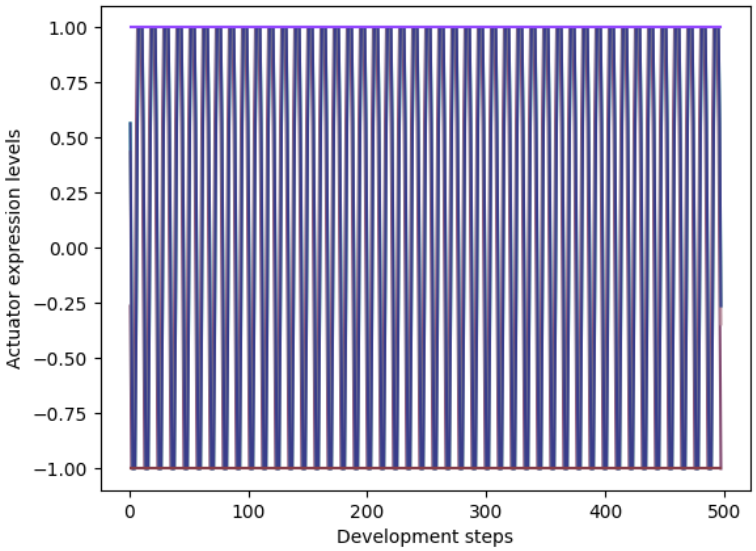


Fig.52. *Co-evolved GRN actuator levels for 'Walker-v0'*

The best-performing robot for the Carrier-v0 task also shows effective behaviour and actuator cycles: **See Appendix F.3**

Statistical Analysis

A Mann-Whitney U test was performed to compare the performance between the co-evolved GRN and PPO+GA. This test produced p-values of 0.006993 for the Walker-v0 and Carrier-v0 tasks.

These p-values are less than the standard 0.05 significance threshold, meaning the Null Hypothesis can be rejected. Therefore, there is a statistically significant difference between the performance of the co-evolved GRN and PPO+GA.

Since the co-evolved GRN's median performance is less than the PPO+GA, this test shows statistically that the co-evolved GRN performed worse than PPO+GA.

See Appendix G.3 for box plot comparison charts

Diversity Analysis

While the co-evolved GRN robots were able to produce effective behaviour, it was observed that the diversity of robot shapes decreased towards the end of each experiment. To illustrate this, the best 20 individuals at the end of an experiment were visualised:

Fig.53. *The best 20 robots in both tasks at the end of an experiment. Walker-v0 is shown (left) and Carrier-v0 (right)*

This visualisation shows that for both tasks, there is a lack of diversity at the end of the experiment, as most of the best robots have the same shape.

Quantitative diversity analysis was also conducted, which shows a similar drop in diversity: **See Appendix H**

4.2.8 Conclusions

The results indicate that a GRN can co-evolve the shape and controller of soft-bodied VSRs, and my developmental approach can produce a variety of robot designs. **See Appendix I for the best co-evolved robots**

However, the new approach did not perform as well as the conventional PPO+GA and achieved lower median scores in each task. The analysis indicates that the diversity of different robot designs decreases over time. The algorithm appears to focus on finding a good robot design before optimising the controller for that design. This suggests that the algorithm selects sub-optimal robot designs, which causes it to converge prematurely.

Evaluation and Reflections

Research and Experiment Approach

Overall, I believe that this project has addressed the original research question:

- *Can GRNs be evolved to effectively control soft robots on locomotion and object manipulation tasks?*

This project has explored and answered this question by conducting experiments across three control tasks (two locomotion tasks and one object-manipulation task).

I ensured that my experimentation technique was robust by conducting multiple samples and setting up a fair comparison by evaluating both controllers on the same control task using EvoGym.

Additionally, I applied best practices for conducting research experiments by using different random seeds to ensure that each sample was independent, and I documented my approach, including the experiment parameters, so that the experiments could be replicated. I also performed statistical analysis using the Mann-Whitney U Test to test the statistical significance of the results at each iteration.

In **Chapter 4**, I considered different strategies to improve the GRN's performance, including implementing an alternative evolutionary algorithm and a GRN developmental model to co-design the robot's controller and shape.

Software Implementation and Testing Approach

For implementing software, I followed an iterative development methodology, which involved modifying the code to implement new features, testing the code and evaluating the model using EvoGym during experiments. This approach was chosen rather than a waterfall model since the approach to improve the GRN could not be known until after experiments were conducted on the prior iteration.

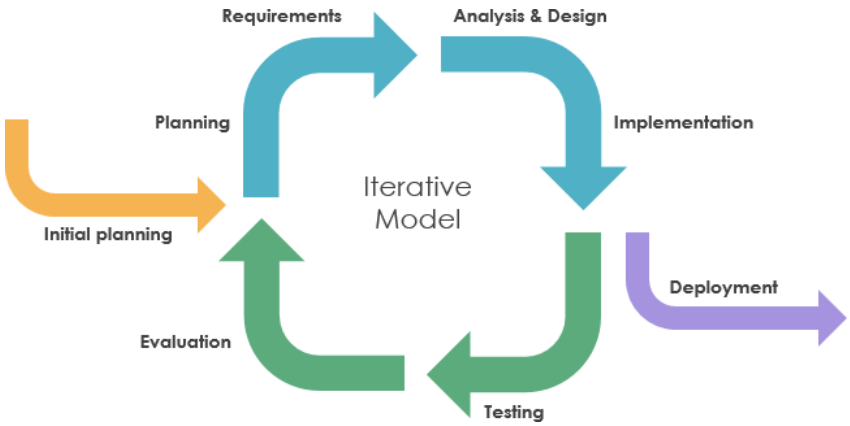


Fig.54. *Iterative Model (Visual Paradigm, 2024)*

I also applied different testing methods, such as Unit Testing and System Testing, to ensure that my implemented code was error-free and produced the expected results. Using these methods allowed me to identify errors early in the development lifecycle.

To support these methods, I used GitHub to push commits from PyCharm to back up my work during each iteration.

Reflections

Overall, I am pleased with how I conducted the project and my achievements. I followed my project plan; however, I did have to modify the plan to adapt to the circumstances.

For example, I initially intended to compare the GRN against an ANN approach (Tanaka, Aranha, 2022), but unfortunately, I could not get their code to run, so I chose to replicate PPO instead.

Overall, the biggest challenge for this project was managing time, as many experiments took multiple days to complete, and I needed a sufficient number of samples for each task to perform statistical analysis. I accommodated this by reinstalling EvoGym on a remote PC to allow experiments to run overnight whilst also running experiments overnight on my laptop.

There are a couple of things that I would have done differently in this project:

- I would have decided on a control task and begun the replication experiments sooner in the project. I did not anticipate that each experiment would take several days, which meant it was challenging to collect enough samples to conduct a fair statistical comparison for the second iteration.

- I would have requested a laboratory PC earlier in the project to have more time and resources to conduct more experiments.

Conclusions

This project replicated and applied a Gene Regulatory Network to evolve complex, soft-bodied robots for locomotion and object-manipulation tasks. An iterative methodology was followed both for researching different approaches and implementing these to refine the model.

The findings show that a GRN model can be evolved to effectively control VSRs when using an evolutionary algorithm that employs crossover and Tournament Selection. This approach was extended to implement a GRN developmental model which co-evolves the robot's shape and controller. These results could be improved by preserving the diversity of robot shapes.

Overall, the project's aims and objectives have been achieved:

- For Objective 1, this project replicated the work of (Bhatia et al., 2021) using a conventional PPO model in **Section 2.2**.

- For Objective 2, this project implemented (Watson et al., 2014)'s GRN model and an evolutionary algorithm in **Chapter 3** using Python.

- For Objective 3, this project conducted experiments to compare the GRN with the existing PPO model for controlling fixed robot designs in **Sections 3.3 and 4.1.4** and for co-designing robots in **Section 4.2.4**

- For Objective 4, non-parametric statistical tests were conducted following experiments to measure the statistical significance of results. While I was able to collect a sufficient number of samples for most experiments, in the second refinement stage, I was only able to collect 3 samples for the PPO+GA approach, as experiments took multiple days.

- For Objective 5, different approaches were successfully implemented to improve the GRN's performance, including an alternative evolutionary algorithm and a method for co-evolving the robots' brain and body.

- For Objective 6, the GRN model was applied to more complex tasks, such as the UpStepper-v0; however, due to time constraints, I was not able to apply this task in the second refitment iteration. I would have liked to utilise more of EvoGym's tasks; however, I did not have sufficient computational resources or time to achieve this.

Aims	
1	Explore the effectiveness of Gene Regulatory Networks as a suitable model for controlling Voxel-Soft-bodied Robots across different simulated control tasks and environments.
2	Compare and evaluate the performance of robots evolved using a GRN model with a conventional model across various control tasks and study the capabilities and behaviours of each model.
3	Explore methods to improve the performance of the GRN model.
Objectives	
1	Replicate the findings of a relevant research paper where Voxel-based Soft Robots are controlled with a conventional model. This will act as the baseline model, which will be used for comparing against the GRN-inspired model.
2	Design and implement a GRN model, which can be evolved to control of a soft-bodied agent to achieve particular tasks.
3	Conduct experiments to compare the GRN with the existing baseline model on various agent control tasks.
4	Use relevant statistical methods to compare and analyse the evolutionary performance of the robots in each experiment.
5	Investigate how varying the GRN model's hyperparameters or using a different GRN model could improve the evolved robots' performance.
6	If the GRN can successfully control robots to complete a simple task, conduct further experiments on the GRN model to assess its ability to control robots on more complex control tasks and environments

Fig.55. *Project Aims and Objectives*

There are many interesting ways to expand on this project's work:

- In **Section 4.2**, the robot's shape and controller were co-evolved using a decentralised GRN. Future projects could adapt this to explore how GRNs can evolve their sensory inputs so that robots can adapt sensors suited for the task.

- The current EvoGym tasks require robots to learn one behaviour, such as running for the locomotion tasks. (Watson et al., 2014) showed that recurrent GRNs have memory capabilities and can learn different patterns. Future work could use EvoGym to design a task that combines two different environments and explores

whether this GRN model can enable robots to switch between different patterns of behaviour suited to each environment.

- This project utilised an abstract recurrent neural network GRN model to control and design VSRs. Future work could compare this abstract model with a more biologically realistic model to evaluate each model's capabilities.

References

Asr, N.R. & Majd, V.J. 2013, "A new artificial genetic regulatory network model and its application in two dimensional robot control", *International Journal of Information and Electronics Engineering*, vol. 3, no. 5, pp. 461.

Aston, E., Channon, A., Belavkin, R.V., Gifford, D.R., Krašovec, R. & Knight, C.G. 2017, "Critical mutation rate has an exponential dependence on population size for eukaryotic-length genomes with crossover", *Scientific reports*, vol. 7, no. 1, pp. 15519.

Bhatia, J., Jackson, H., Tian, Y., Xu, J. & Matusik, W. 2021, "Evolution gym: A large-scale benchmark for evolving soft robots", *Advances in Neural Information Processing Systems*, vol. 34, pp. 2201-2214.

Cheney, N., MacCurdy, R., Clune, J. & Lipson, H. 2014, "Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding", *ACM SIGEVOlution*, vol. 7, no. 1, pp. 11-23.

Cussat-Blanc, S., Harrington, K. & Banzhaf, W. 2019, "Artificial Gene Regulatory Networks—A Review", *Artificial Life*, vol. 24, no. 4, pp. 296-328.

Delgado, F.M. & Gómez-Vela, F. 2019, "Computational methods for Gene Regulatory Networks reconstruction and analysis: A review", *Artificial Intelligence in Medicine*, vol. 95, pp. 133-145.

Hassanat, A.B. & Alkafaween, E. 2017, "On enhancing genetic algorithms using new crossovers", *International Journal of Computer Applications in Technology*, vol. 55, no. 3, pp. 202-212.

Hiller, J. & Lipson, H. 2014, "Dynamic simulation of soft multimaterial 3d-printed objects", *Soft robotics*, vol. 1, no. 1, pp. 88-101.

J. Hiller & H. Lipson 2012, "Automatic Design and Manufacture of Soft Robots", *IEEE Transactions on Robotics*, vol. 28, no. 2, pp. 457-466.

Joachimczak, M., Suzuki, R. & Arita, T. 2014, "Fine Grained Artificial Development for Body-Controller Coevolution of Soft-Bodied Animats", - *ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems*, pp. 239.

Kriegman, S., Cappelle, C., Corucci, F., Bernatskiy, A., Cheney, N. & Bongard, J.C. 2017, "Simulating the evolution of soft and rigid-body robots", *Proceedings of the genetic and evolutionary computation conference companion*, pp. 1117.

Liu, Z., Peng, K., Han, L. & Guan, S. 2023, "Modeling and control of robotic manipulators based on artificial neural networks: a review", *Iranian Journal of Science and Technology, Transactions of Mechanical Engineering*, vol. 47, no. 4, pp. 1307-1347.

McKnight, P.E. & Najab, J. 2010, "Mann-Whitney U Test", *The Corsini encyclopedia of psychology*, , pp. 1.

Medvet, E., Bartoli, A., De Lorenzo, A. & Fidel, G. 2020a, "Evolution of distributed neural controllers for voxel-based soft robots", *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 112.

Medvet, E., Bartoli, A., De Lorenzo, A. & Seriani, S. 2020b, "2D-VSR-Sim: A simulation tool for the optimization of 2-D voxel-based soft robots", *SoftwareX*, vol. 12, pp. 100573.

Moreira, A.L. & Rennó-Costa, C. 2021, "Evolutionary Strategies Applied to Artificial Gene Regulatory Networks", *bioRxiv*, , pp. 2021.09. 28.462218.

Mukesh Patel, Vasant Honavar & Karthik Balakrishnan 2001, *Advances in the Evolutionary Synthesis of Intelligent Agents*, The MIT Press.

Nadizar, G., Medvet, E., Nichele, S. & Pontes-Filho, S. 2023, "An experimental comparison of evolved neural network models for controlling simulated modular soft robots", *Applied Soft Computing*, vol. 145, pp. 110610.

Peter, I.S. & Davidson, E.H. 2011, "Evolution of gene regulatory networks controlling body plan development", *Cell*, vol. 144, no. 6, pp. 970-985.

Pigozzi, F. & Medvet, E. 2022, "Evolving modularity in soft robots through an embodied and self-organizing neural controller", *Artificial Life*, vol. 28, no. 3, pp. 322-347.

Prügel-Bennett, A. 2004, "When a genetic algorithm outperforms hill-climbing", *Theoretical Computer Science*, vol. 320, no. 1, pp. 135-153.

Razali, N.M. & Geraghty, J. 2011, "Genetic algorithm performance with different selection strategies in solving TSP", *Proceedings of the world congress on engineering*International Association of Engineers Hong Kong, China, , pp. 1.

Sanchez, S. & Cussat-Blanc, S. 2014, "Gene regulated car driving: using a gene regulatory network to drive a virtual car", *Genetic Programming and Evolvable Machines*, vol. 15, pp. 477-511.

Sims, K. 1994, "Evolving virtual creatures", *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*Association for Computing Machinery, New York, NY, USA.

Singh, B., Kumar, R. & Singh, V.P. 2022, "Reinforcement learning in robotic applications: a comprehensive survey", *Artificial Intelligence Review*, vol. 55, no. 2, pp. 945-990.

Sushant Gaurav 2023, , *Advantages and Disadvantages of OOP*.

Tanaka, F. & Aranha, C. 2022, "Co-evolving morphology and control of soft robots using a single genome", *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*IEEE, , pp. 1235.

Visual Paradigm 2024, *What is a Software Process Model?*. Available: https://www.visual-paradigm.com/guide/software-development-process/what-is-a-software-process-model/.

Watson, R.A., Wagner, G.P., Pavlicev, M., Weinreich, D.M. & Mills, R. 2014, "The evolution of phenotypic correlations and “developmental memory”", *Evolution*, vol. 68, no. 4, pp. 1124-1138.

Wong, C., Yang, E., Yan, X. & Gu, D. 2018, "Autonomous robots for harsh environments: a holistic overview of current solutions and ongoing challenges", *Systems Science & Control Engineering*, vol. 6, no. 1, pp. 213-219.

Zahadat, P., Christensen, D.J., Katebi, S. & Stoy, K. 2013, "Sensor-coupled fractal gene regulatory networks for locomotion control of a modular snake robot", *Distributed Autonomous Robotic Systems: The 10th International Symposium*Springer, , pp. 517.

Appendices

A. Project Plan

UG Project Plan
CSC-30014

Project Overview and Description

Student Name: Nathaniel Brookes
Student Username: x5f33
Student Number: 21011025
Degree Title: Computer Science BSc (Single Honours)
Supervisor Name: Dr Alastair Channon
Project Title: Co-evolving the Body and Brain of Soft Modular Robots Using a Gene Regulatory Network

Please provide a brief Project Description:

This research-based project will investigate how the shape and controller of soft-bodied modular robots can be co-evolved to perform control tasks, such as navigating terrain in a 2D virtual environment. The robots are a type of Voxel-Soft-bodied Robot (VSR), where the agent is comprised of many 2D voxels that can expand and contract.

This project will utilise a Gene Regulatory Network (GRN) to co-evolve the robots. This model is inspired by how the genes in our cells regulate the expression of other genes, and this process can dynamically influence cell behaviour. Previous work has shown that Gene Regulatory Networks can be used to design and control simulated agents. (Cussat-Blanc, Harrington & Banzhaf, 2019).

Traditionally, models such as Feed-forward Neural Networks have been used to control and design VSRs. However, agents controlled using a GRN can dynamically adapt to changes in their environment, and GRNs have been shown to explore the search space more effectively. (Yao et al., 2016)

This project will compare a conventional model with a GRN model, which will be designed and implemented using existing simulation software. Both models will be analysed to study how each evolved population performs over time, and following initial experiments, the GRN model will be refined, and different GRN models will be considered to improve the results.

What are the aims and objectives of the Project?

Aims:

- To explore the effectiveness of Gene Regulatory Networks as a suitable model for co-evolving the shape and controller of Voxel-Soft-bodied Robots across different simulated control tasks and environments.
- To compare and evaluate the performance of robots evolved using a GRN model with a conventional model across various control tasks and study the capabilities and behaviours of each model.
- To improve the performance of the GRN model.

Objectives:

1. Replicate the findings of a relevant research paper where populations of Voxel-based Soft Robots are co-evolved using a conventional model, such as a feed-forward neural network. This will act as the baseline model, which will be compared with a GRN-inspired model.

2. Design and implement a Gene Regulatory Network model, which can be evolved to control the shape and controller of a soft-bodied agent to achieve particular tasks.

3. Conduct experiments to compare the GRN with the existing baseline model on various agent control tasks.

4. Use relevant statistical methods to compare and analyse the evolutionary performance of the robots in each experiment.

5. Investigate how varying the GRN model's hyperparameters or using a different GRN model could improve the evolved robots' performance.

6. If the GRN can successfully evolve robots to complete a simple task, conduct further experiments on the GRN model to assess its ability to design robots which can handle more complex control tasks and environments.

Please provide a brief overview of the key literature related to the Project:

- **Artificial Gene Regulatory Networks—A Review**

  In Chapter 5 of this review, the authors discuss many previous examples of Gene Regulatory Network models which have been applied to design and control agents in a wide range of tasks. Examples of GRN models include cellular automata, Boolean networks, and cellular development models. This review highlights that most agent-control GRNs are encoded as Ordinary Differential Equations.

  (Cussat-Blanc, Harrington & Banzhaf, 2019)

- **Evolution Gym: A Large-Scale Benchmark for Evolving Soft Robots**

  This paper introduces Evolution Gym - an open-source benchmarking tool specifically developed to help researchers compare and develop new algorithms for co-evolving the shape and controller of 2D modular soft robots. Evolution Gym has various locomotion and object manipulation tasks and simulated environments, which can be used to evaluate the abilities of the evolved robots.

  (Bhatia et al., 2021)

  - Evolution Gym is open-source and available on GitHub: https://github.com/EvolutionGym/evogym

- **Co-evolving morphology and control of soft robots using a single genome**

  This study introduces a new method for co-evolving the design and controller of robots using the hyperNEAT algorithm. This algorithm generates two neural networks, one for designing the robot's shape and another for controlling its movements. In contrast to previous studies, where the evolution of the design and controller are usually treated as separate processes, this paper co-evolves the design and controller of the robot together using a single genome. This algorithm is applied to four control tasks and evaluated using Evolution Gym. The results of this study show that their co-design algorithm can produce robots that can achieve the four control tasks.

  (Tanaka, Aranha, 2022)

  - This paper will be used as the baseline model, which can be compared using Evolution Gym with a developed GRN model.

  - This paper provides the source code for its experiments on GitHub: https://github.com/fhtanaka/SGR

Project Processes and Methods

Please provide a brief overview of the Methodology to be used in the Project (inc. an overview of best practice within the Methodology):

Software Development & Research Methodologies

The aims and objectives of this project will be achieved using an iterative development methodology, which is a typical approach used in software engineering (InterviewBit, 2022). The project will involve three key deliverables: (1) Replicating and verifying existing results, (2) The first implementation of the GRN model, and (3) An improved implementation of the GRN.

- Deliverable 1: Replicating and Verifying Existing Results

  - This will involve downloading and running the code for the baseline model using the Evolution Gym simulator.

  - The experiments conducted in the paper will be verified by testing the code and comparing that the results match with the paper. The paper evaluates four tasks: Walker-v0, ObstacleTraverser-v1, Climber-v2, and Thrower-v0. These are baseline control tasks already implemented in Evolution Gym.

  - To demonstrate that the replication results match those in the paper, the fitness scores and curves of the evolved robots will be measured during the experiments for each task and compared with the fitness scores and curves shown in the paper.

- Deliverable 2: The First Implementation of the GRN

  - This deliverable will involve designing and implementing a GRN model in Python

using the Evolution Gym simulator. Several GRN models exist for evolving the robot's shape and controller, such as a linear genome used in (Joachimczak et al., 2013), a recurrent neural network used in (Joachimczak, Suzuki & Arita, 2014), and fractal GRNs described in (Bentley, 2004).

- After implementing the GRN, the same four control tasks used in Deliverable 1 will be applied to test the new model's abilities. The fitness scores and scores generated from these experiments will be compared against the results of the baseline model in Deliverable 1.

- To analyse the performance results of the GRN, standard Python graph-making libraries (matplotlib.pyplot and pandas) will be used to plot the fitness curves for each experiment. The fitness score will be shown on the y-axis, and the generations will be shown on the x-axis.

- To compare the GRN's performance with the baseline model's performance, the same Python libraries will be used to plot the fitness score curves for both models on the same graph.

- This deliverable will also involve running a statistical comparison of the GRN and the baseline model's performance to compare the performance of the models.

- Deliverable 3: An Improved Implementation of the GRN

  - For this deliverable, the performance of the GRN across the different control tasks experiments conducted in the previous deliverable will be evaluated. This will determine which tasks the current GRN model struggles with and inform what parts of the GRN model could be re-designed to improve the results.

  - A new GRN model will be designed and implemented, and the same graphing and statistical methods used in Deliverable 2 will be run to compare and evaluate the performance of the new GRN model against the other models.

Will any special Data Collection Methods will be employed (e.g card sorts, questionnaires, simulations, …)?

Simulation data will be collected using the simulation software, including the fitness scores of robots in each experiment. This data will be processed and visualised using Python libraries to generate graphs.

Briefly describe how you will ensure your project is in line with BCS Project Guidelines (BSc Computer Science Single Honours Students only)?

- My project involves developing a GRN model, which I will implement using Python within an existing simulation software framework. Therefore, I will be "undertaking practical work of some sort using computing/IT technology".
- I have devised a project plan and timeline by creating a Gannt chart, which I will use to keep track of my project and revise if necessary.
- My project will involve creating a report (dissertation) containing my research aims, objectives, and a review of previous work. In my report, I will describe all the software tools I will use and my reasoning for choosing them. I will critically evaluate the results that I will obtain through the experiments. I will also evaluate my work and consider where lessons can be learnt and things that I could have done differently.
- My project report will include references.

Time and Resource Planning

Will Standard Departmental Hardware be used? YES

If NO please outline the Hardware/Materials to be used:

Will Software which is already available in department be used? YES

If NO please outline the Software to be used including how any necessary licences will be obtained:

Will the project require any Programming? YES

If YES please list the (potential) Programming Languages to be used (including any IDEs and Libraries you may make use of):

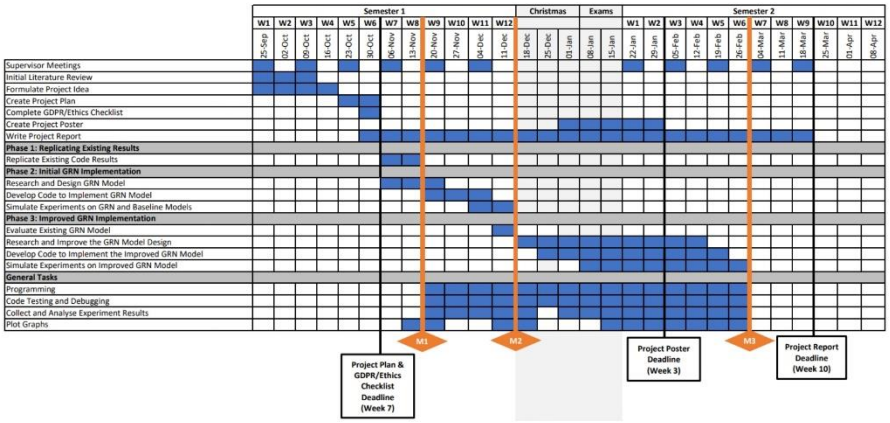
- Python (For simulating the baseline and GRN models, conducting evolutionary experiments, and for data analysis and visualisation)
  - IDE: **PyCharm** (Using Python 3.7 virtual environment)
  - Libraries:
    - Standard Python libraries (e.g. random, os)
    - **EvoGym** Simulator (For simulating and evaluating the baseline and GRN models)
    - **matplotlib.pyplot** and **pandas** (For data analysis and visualisation)

Table of Risks (*if non Standard Hardware and/or Software to be used please also include backup options/ contingency plans here)*:

Risk-id Description	Probability/Likelihood of occurring	Best practice prevention measures	Remedy
Loss of work (E.g. Theft of laptop or accidental deletion of code)	**Low**	**Use GitHub to upload code regularly.** **Backup files using OneDrive and an external hard drive.**	**Restore files from GitHub, OneDrive and external hard drive where necessary.**
It is too difficult to integrate a GRN model within the existing open-source simulator.	**Low-Medium** The EvoGym has been specifically developed for researchers to benchmark their algorithms; therefore, there should be capabilities to integrate a new GRN model.	**Find appropriate classes and methods in the simulation library which can be extended or adapted to implement a GRN.**	**Contingency:** Explore other software libraries which could be used instead, such as 2D-VSR-Sim. If those do not work, write code for simulating and evolving agents from scratch (using an existing 2D physics engine).
The computational demands for evolving 2D modular robots are too high for my laptop.	**Medium** Simulating 2D Voxel-based Soft Robots using a simple Spring-Mass physics implementation is not too computationally demanding. However, evolving a larger population of these robots to achieve a complex task could take more time to compute than what is reasonable.	**Start with a smaller control task or a simpler environment.** **Carefully inspect the model's hyperparameters so that it can run on the laptop.**	**Reduce the complexity of the task. For example, evolve robots on a flatter terrain.** **Use different hyperparameters for the model, such as reducing the number of genes in the GRN.**
The open-source software does not run on my machine (E.g. not compatible or buggy)	**Medium** Most research software is open source, and the code may not have been	**Test code early before continuing.** **Ensure the development**	**Contingency:** Explore other software libraries which could be used

	tested properly.	**environment is set up correctly before using the software.**	instead, such as 2D-VSR-Sim. If those do not work, write code for simulating and evolving agents from scratch (using an existing 2D physics engine).
Bugs in the code that I will write	**Medium-High**	**Test code regularly.** **Use the Git version-control system so that I can track changes and roll back code.**	**Use the IDE in-built debugging tool to help identify errors.** **Roll back code to a previous version if necessary.**

Gantt Chart (must include milestones and deliverables):



References

Please include a list of References used in this Plan (using Harvard reference style):

André L. L. Moreira & César Rennó-Costa 2021, "Evolutionary Strategies Applied to Artificial Gene Regulatory Networks", *bioRxiv,* , pp. 2021.09.28.462218.

Bentley, P.J. 2004, "Evolving beyond perfection: An investigation of the effects of long-term evolution on fractal gene regulatory networks", *Biosystems,* vol. 76, no. 1-3, pp. 291-301.

Bhatia, J., Jackson, H., Tian, Y., Xu, J. & Matusik, W. 2021, "Evolution gym: A large-scale benchmark for evolving soft robots", *Advances in Neural Information Processing Systems,* vol. 34, pp. 2201-2214.

Cussat-Blanc, S., Harrington, K. & Banzhaf, W. 2019, "Artificial Gene Regulatory Networks—A Review", *Artificial Life,* vol. 24, no. 4, pp. 296-328.

InterviewBit 2022, , *Iterative Model – Software Engineering*. Available: https://www.interviewbit.com/blog/iterative-model/.

Joachimczak, M., Kowaliw, T., Doursat, R. & Wróbel, B. 2013, "Controlling development and chemotaxis of soft-bodied multicellular animats with the same gene regulatory network", *Artificial Life Conference Proceedings*MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info ..., , pp. 454.

Joachimczak, M., Suzuki, R. & Arita, T. 2014, "Fine Grained Artificial Development for Body-Controller Coevolution of Soft-Bodied Animats", - *ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems*, pp. 239.

Tanaka, F. & Aranha, C. 2022, "Co-evolving morphology and control of soft robots using a single genome", *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*IEEE, , pp. 1235.

Yao, Y., Storme, V., Marchal, K. & Van de Peer, Y. 2016, "Emergent adaptive behaviour of GRN-controlled simulated robots in a changing environment", *PeerJ,* vol. 4, pp. e2812.

Submission Date: 05/11/2023

PLEASE NOTE THAT SHOULD YOUR PROJECT UNDERGO ANY MAJOR CHANGES FOLLOWING THE SUBMISSION OF THIS PLAN YOU ARE EXPECTED TO SUBMIT AN UPDATED PLAN WHICH ACCURATELY REFLECTS YOUR PROJECT.

B. Project GDPR and Ethics Checklist

Computer Science CSC-30014 Project GDPR and Ethics Checklist 2023-24

STUDENT NAME: Nathaniel Brookes

STUDENT NUMBER & E-MAIL: 21011025 x5f33@students.keele.ac.uk

PROJECT TITLE: Co-evolving the Body and Brain of Soft Modular Robots Using a Gene Regulatory Network

SUPERVISOR NAME: Dr Alastair Channon

Complete all the questions below and electronically sign and date at the end. Ask your supervisor to check the responses and to also electronically sign and date below. You should submit this completed checklist to the KLE drop-box provided. Please retain your own copy of the completed checklist as all end of project reports/dissertations must provide a copy of this completed checklist in the Appendices.

If a Computer Science Final-Year Project Ethical Review Application Form has also had to be completed a copy of the final ethical approval certificate must also be included with your Project Plan document in your final report/dissertation Appendices.

GDPR Check

Does your project involve the use or collection of "personal data" for which permission will not have been explicitly granted? (Before answering, please read and carefully consider the GDPR Check Guidance at the end of this form).	No

Ethics Check

Will your project involve the use of human participants or capturing human data? (Before answering, please read and carefully consider the Significant Ethical Concern Checklist, below). Please Note: software evaluation by any other person counts as human participation. If you ask people their opinions about software or use their data in any way, you need to seek ethical approval first. If you answered "Yes" to this question you must discuss your plans with your supervisor and, by end of week 12 of Semester 1, complete the Computer Science Final-Year Project Ethical Review Application Form and prepare a Participant Information Sheet and Consent Form using the templates that can be found at the end of that application form.	No

Significant Ethical Concern Checklist

Could the project expose the participants or the project student to images and/or information that they might find distressing (e.g. pictures or descriptions of injuries,	No

symptomatic health conditions, or atrocities, or pictures or descriptions of tumours or cancerous cells, or creatures in distress)?	
Does the project involve deception of the participants?	No
Could the project uncover information about identifiable individuals that could cause embarrassment or distress to one or more of those individuals (e.g. evidence of illegal or unethical behaviour, such as fraud or illegal drug use or a personal revelation)?	No
Could the project cause pain, discomfort or risk to the participants and/or the project student?	No
Will the project involve participants who are vulnerable in any way? (e.g. participants who are under 18, or who are mentally or physically impaired, or participants who may feel under pressure to participate.)	No
Does the project involve recall or discussion of personal or sensitive memories?	No
Does the project involve a significant risk of participants later regretting taking part?	No
Does the project involve procedures which are likely to provoke interpersonal or inter-group conflict?	No

If the answer to any of the Significant Ethical Concern Checklist questions is "Yes" (or you think "Maybe") you must discuss your project and aims with your supervisor and with the CSC-30014 Module Co-ordinator to assess whether an appropriate level of ethical scrutiny might be required via the completion of the *Faculty of Natural Sciences (non-Psychology) Research Ethics Application Form*.

GDPR Check Guidance:

Personal data includes any and all of: names, addresses, emails, phone numbers, bank details, employment details, IP addresses, date of birth, medical or health data, images, video or audio recordings.

If you answered "Yes" you must not proceed with your project.

It is illegal under European GDPR legislation to make use of personal data without explicit permission. Discuss your project with your supervisor and revise your plans to ensure you do not risk illegal use of personal data.

Note. Even if personal data is publicly available on the Internet, it must not be used without permission. (Also note that you cannot contact individuals to request permission to use their on-line data without prior ethical approval to do so).

It is strongly recommended that you either:

1. use non-personal data for your project, or
2. use existing, well-established, publicly available databases or data repositories, for example: https://archive.ics.uci.edu/ml/index.php, https://physionet.org/ or https://www.kaggle.com/ etc. (A list of acceptable data repositories is maintained on the CSC-30014 KLE pages.) You might also see, for example, https://blog.scrapinghub.com/web-scraping-gdpr-compliance-guide) for further information.

Continued on the following page ...

I confirm that the responses are correct and that the project, as proposed, is GDPR compliant and that ethical approval will be sought if required and that any work requiring ethical approval will not take place unless ethical approval has been granted. If, during the course of the project work, any of the information supplied on this checklist changes substantially a new checklist will need to be completed and then brought to the attention of the School's Ethics Advisory team.

Signed (Student)	Date:
MR NATHANIEL BROOKES	05/11/2023

I confirm that I have read the form and that the project, as proposed, is GDPR compliant and that, to the best of my knowledge, the ethical information is correct.

Signed (Supervisor)	Date:
DR ALASTAIR CHANNON	05/11/2023

Electronically typed signatures and dates *are* acceptable.

C. Project Poster

D. Unit Test Cases for Software Implementation

D.1: Tests for Replicated GRN Model

Unit Test Plan

Test No.	Function and Justification	Test Data	Expected Result
1.1	calculate_fitness() To check that the fitness calculation is correct.	***Scenario 1:*** Two arrays will be passed with the same values. v1 = [1,1,1] v2 = [1,1,1]	The result should compute the dot product and output: (1*1) + (1*1) + (1*1) = **3**
		Scenario 2: Two arrays will be passed with inverse values. v1 = [1,1,1] v2 = [-1,-1,-1]	The result should compute the dot product and output: (1*-1) + (1*-1) + (1*-1) = **-3**
1.2	develop() To check that the program can correctly develop the robot's genotype into a phenotype.	***Scenario 1:*** Dummy values for *gene_potentials* and an empty *interaction_matrix* will be generated and passed into develop() with a developmental time step of 1. gene*_potentials* = [0, 0.5, 1] *interaction_matrix* = [0, 0, 0, 0, 0, 0, 0, 0, 0]	The result should reduce the *gene_potentials* by 20% since this is the degradation rate and output: 0*0.8 = 0, 0.5*0.8 = 0.4, 1*0.8 = 0.8 = **[0, 0.4, 0.8]**
		Scenario 2: The same values for gene_potentials will be used; however, the *interaction_matrix* will be changed to add one positive connection weight from gene 3 → gene 2. gene*_potentials* = [0, 0.5, 1] *interaction_matrix* = [0, 0, 0, 0, 0, 1, 0, 0, 0]	The result should increase the value of gene 2 and clamp the output between -1 and 1. For gene 2, this should calculate: 0.5*0.8 = 0.4 0.4 + tanh(1) = 1.16152 Clamped output

			= **1**
1.3	mutate() To check that the program correctly mutates the *gene_potentials* and *interaction_matrix* arrays.	An empty gene_*potentials* and *interaction_matrix* will be created. gene_*potentials* = [0, 0, 0] *interaction_matrix* = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	The result should add random mutations to one randomly chosen element within *gene_potentials* and *interaction_matrix*

Unit Test Results

Test No.	Results	Pass / Fail

1.1	***Scenario 1:***
	
	`v1 = [1 1 1]`
	`v2 = [1 1 1]`
	
	`Fitness Result: 3`
	
	`Process finished with exit code 0`
		Pass

	Scenario 2:
	
	`v1 = [1 1 1]`
	`v2 = [-1 -1 -1]`
	
	`Fitness Result: -3`
	
	`Process finished with exit code 0`
		Pass

1.2	***Scenario 1:***
	
	`gene_potentials = [0.  0.5 1. ]`
	`interaction_matrix = [0 0 0 0 0 0 0 0 0]`
	
	`Phenotype: [0.  0.4 0.8]`
	
	`Process finished with exit code 0`
		Pass

	Scenario 2:
	
	`gene_potentials = [0.  0.5 1. ]`
	`interaction_matrix = [0 0 0 0 0 1 0 0 0]`
	
	`Phenotype: [0.  1.16159416 0.8  ]`
	
	`Process finished with exit code 0`
		Fail
		
		(See below for details)

1.3	`gene_potentials = [0. 0. 0.]`
	`interaction_matrix = [0. 0. 0. 0. 0. 0. 0. 0. 0.]`
	
	`Mutated gene_potentials: [ 0.  0.  -0.0245884]`
	`Mutated interaction_matrix: [0.  0.  0.  0.  0.  0.`
	` 0.00660449 0.  0. ]`
	
	`Process finished with exit code 0`
		Pass

Failed Unit Tests

Test No.	Reason for Failure

1.2 (Scenario 2)	This test failed because the result exceeded the maximum value of 1. It was found that the develop() function did not constrain the phenotype values.
	
	This was corrected by adding the following code within develop():
	
	`# Clamps phenotype between -1 and 1`
	`if phenotype[i] < -1:`
	`  phenotype[i] = -1`
	`elif phenotype[i] > 1:`
	`  phenotype[i] = 1`
	
	The test was repeated and produced the correct output:
	
	`gene_potentials = [0.  0.5 1. ]`
	`interaction_matrix = [0 0 0 0 0 1 0 0 0]`
	
	`Phenotype: [0.  1.  0.8]`
	
	`Process finished with exit code 0`

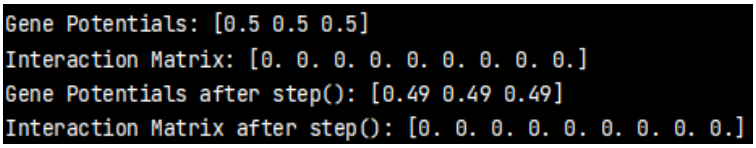
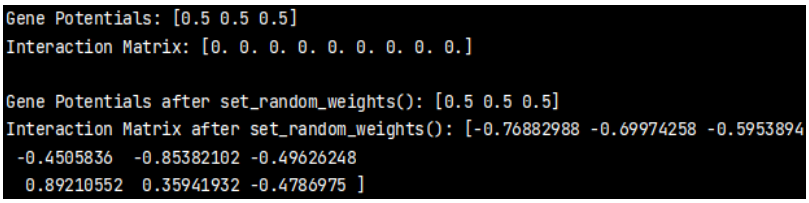
D.2: Tests for Initial GRN Applied to EvoGym

Unit Test Plan

Test No.	Function and Justification	Test Data	Expected Result
2.1	WatsonGRN.step() To check that this method correctly updates the gene potentials.	An instance of WatsonGRN with 3 genes will be created, and the step() method will be run.	The result should reduce the *gene_potentials* by 2% since this is the degradation rate, and there are no regulatory connections.
2.2	WatsonGRN.set_random_weights() To check that this method correctly creates randomised regulatory weights.	An instance of WatsonGRN with 3 genes will be created, and the set_random_weights method will be run.	The result should set each weight with a random value between -1 and 1.
2.3	Robot.set_inputs() To check that robots can input sensor information to its GRN.	An instance of Robot will be created with an environment of 'Walker-v0', and 8 dummy inputs will be passed to the set_inputs method. *inputs* = [0, 1, 1, 0, 1, 0, 0, 1]	The result should show that the first 8 values of robot GRN's gene potentials have been replaced with the dummy inputs.
2.4	Robot.get_actuator_values() To check that the robot can provide its actuator values.	An instance of Robot will be created as previously, and it will be stepped once. Then, the get_actuator_values method will be run.	The result should output the last 10 values of the robot GRN's gene potentials since the action_space is 10.

2.5	GeneticAlgorithm.start() To check that this method can perform a genetic algorithm on a population of robots.	A random population of 10 robots will be created and evaluated in EvoGym.	The result should show that the robots have been sorted according to their fitness. A new population should also be created where the best individual is retained with mutated offspring.

Unit Test Results

Test No.	Results	Pass / Fail
2.1		**Pass**
2.2		**Pass**
2.3		**Pass**

	Sensory Inputs: [0 1 1 0 1 0 0 1] Robot GRN Gene Potentials: [0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5] Robot GRN Interaction Matrix: [0.94660335 0.8676946 0.74544573 ... 0.83958803 0.73 701403 0.95378972] Robot GRN Gene Potentials: after set_inputs(): [0. 1. 1. 0. 1. 0. 0. 1. 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5] Robot GRN Interaction Matrix after set_inputs(): [0.94660335 0.8676946 0.74544573 ... 0.83958803 0.73701403 0.95378972]	
2.4	Robot GRN Gene Potentials: [-0.47749939 -0.34945808 0.72378577 -0.49884952 -0.24329682 1. -0.08912011 1. 1. 1. -0.13744065 -0.47693343 -0.50947375 1. -0.50999982 -0.50845207 1. 1. -0.5096675 1. 1. 1. -0.49673479 -0.50982384 1. 1. -0.50966265 -0.50830427 -0.49242033 1. 1. -0.26402912 -0.48582719 1. -0.50799013 0.08425939 -0.07250007 -0.19274867 -0.37702356 0.2073926 1. 0.503171 1. 0.94777174 1. -0.48025107 1. -0.11519421 -0.10430521 -0.50998592 0.34660233 1. -0.50966621 0.46832104 -0.50949155 -0.50997478 1. 1. -0.49757512 1. 1. -0.42876019 1. -0.50822759 1. 0.30669432 1. 1. 1. -0.50998465 -0.35703174 1. -0.5057491 -0.33205194 1. 0.34306091 -0.50966241 1. 0.8172342 -0.43571204 -0.50927038 1. 0.17056573 -0.22713742 1. 1. -0.21272972 0.67164596 1. -0.50999214 -0.50699273 1. -0.50221444 0.58975695 1. -0.50999976 -0.47897977 0.02861817 -0.5038878 1.] Actuator Values: [-0.50699273 1. -0.50221444 0.58975695 1. -0.50999976 -0.47897977 0.02861817 -0.5038878 1.]	**Pass**
2.5	Generation 0 Finished in 15.86 second(s) 1) 0.8777095595426884 [index 5] 2) 0.521498048051932 [index 1] 3) 0.4376520675763298 [index 4] 4) 0.32018479893445195 [index 7] 5) 0.16332268551482743 [index 0] 6) 0.025664678542762942 [index 6] 7) -0.04164292660743024 [index 9] 8) -0.09930762821703581 [index 3] 9) -0.24355103923465146 [index 2] 10) -0.7116326059610275 [index 8] Generation 1 Finished in 16.26 second(s) 1) 0.8777095595426884 [index 0] 2) 0.8691387080109226 [index 6] 3) 0.7215187323851119 [index 2] 4) 0.5725540684698779 [index 4] 5) 0.25420385987413585 [index 3] 6) 0.24883939083057105 [index 1] 7) 0.13247758262077147 [index 8] 8) 0.12028796109255846 [index 9] 9) -0.31439502102371775 [index 7] 10) -0.722759625645933 [index 5]	**Pass**

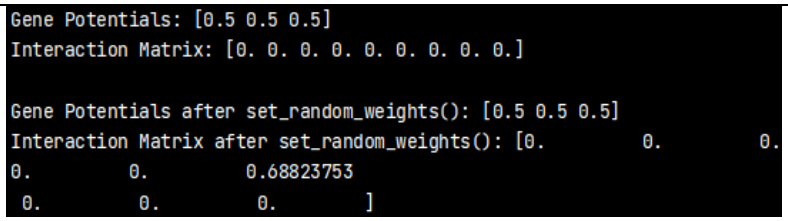
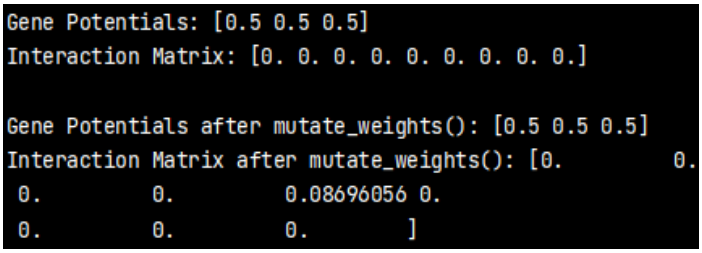
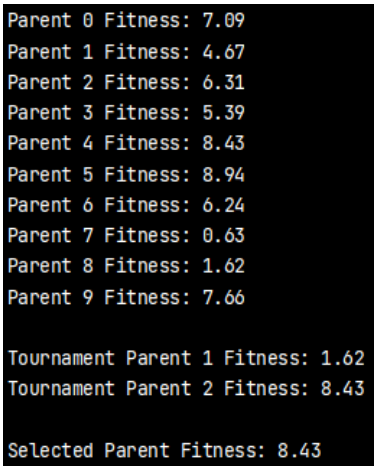
D.3: Tests for Refined GRN

Unit Test Plan

Test No.	Function and Justification	Test Data	Expected Result
3.1	WatsonGRN.set_random_weights() This method was updated; therefore, this test will check that the GRN can correctly create randomised regulatory weights.	An instance of WatsonGRN with 3 genes will be created, and the set_random_weights() method will be run.	The result should set 10% of weights with a random value between -1 and 1.
3.2	WatsonGRN.mutate_weights() To check that the GRN can correctly mutate its regulatory weights.	An instance of WatsonGRN with 3 genes will be created, and the mutate_weights() method will be run.	The result should mutate regulatory weights by adding a random value to 5% of the weights and resetting 1% of the weights to 0.
3.3	GeneticAlgorithm.start() - Tournament Selection This method was updated to introduce Tournament Selection; therefore, this test will check that the code correctly applies Tournament Selection to select the robots.	Ten random robots will be instantiated and given random fitness scores. The code for tournament selection will be run on these ten to execute a tournament of size two.	The result should show that 2 robots were randomly chosen from the population, and the individual with the highest fitness should be selected.

Unit Test Results

Test No.	Results	Pass / Fail
3.1		**Pass**

		
3.2		**Pass**
3.3		**Pass**

D.4: Tests for Co-evolved GRN

Unit Test Plan

Test No.	Function and Justification	Test Data	Expected Result
4.1	RobotVoxel.determine_fate() To check that the decentralised controller matures into the correct material state based on the values of its fate genes.	An instance of Robot will be created, and its GRN controller will be specified with dummy values for its fate genes.	The result should show that the voxel's fate has been updated based on the highest fate gene.

4.2	Robot.develop() This method was added to allow the robot to develop by creating instances of RobotVoxel; therefore, this test will check that the RobotVoxel functions correctly.	An instance of Robot will be created, and the develop() method will run to simulate the developmental process.	The result should generate a robot with multiple voxels connected to each other within the 5x5 design space.

Unit Test Results

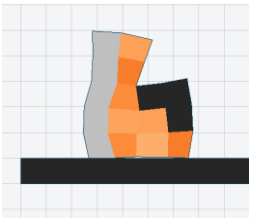
Test No.	Results	Pass / Fail

4.1	***Run 1:***
	
	`Voxel Fate Genes: [1, 0, 0, 0]`
	
	`Voxel Fate = Material Type 0`
	
	Run 2:
	
	`Voxel Fate Genes: [0, 0, 1, 0]`
	
	`Voxel Fate = Material Type 2`
		Pass

4.2	`Traceback (most recent call last):`
	`  File "test.py", line 50, in <module>`
	`  unit_test_1()`
	`  File "test.py", line 27, in unit_test_1`
	`  env = gym.make('Walker-v0', body=robot.get_structure()[0])`
	`  File "C:\Users\Nathaniel\Desktop\PyCharm\evogym\examples\my_examples\coevolve_try\robot.py", line 51, in get_structure`
	`  body[key] = voxel.fate+1`
	`IndexError: index 5 is out of bounds for axis 1 with size 5`
		Fail
		
		(See below for details)

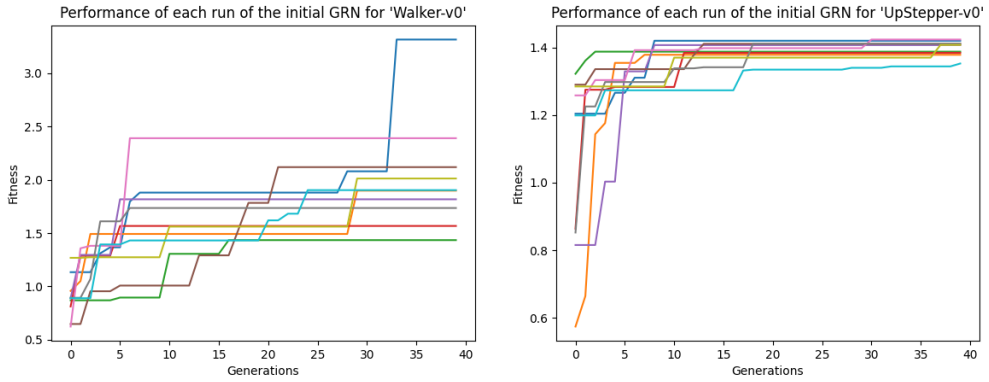
Failed Unit Tests

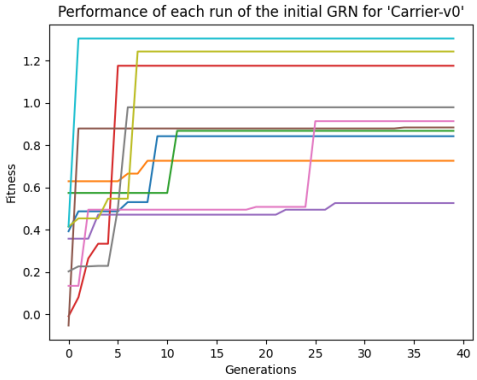
Test No.	Reason for Failure
4.2	This test failed due to a runtime error, which was reported in the get_structure() method of the Robot class. This result was found to be caused by an error within the RobotVoxel class when it checks whether a voxel can divide into an adjacent voxel.

	The code allows voxels to be created at index 5, and since the indices start from zero, this caused the robot to divide beyond the permitted 5x5 design space:
	
	`# Attempt to divide voxel to right`
	`if col+1 <= self.robot.container_shape[0]`
	`    self.robot.voxels[(col+1, row)] = Robo`
	
	This was corrected by reducing the maximum index by 1:
	
	`# Attempt to divide voxel to right`
	`if col+1 <= self.robot.container_shape[0]-1`
	`    self.robot.voxels[(col+1, row)] = RobotV`
	
	The test was repeated, which showed no errors and produced the intended result:
	
	 

E. Fitness Graphs

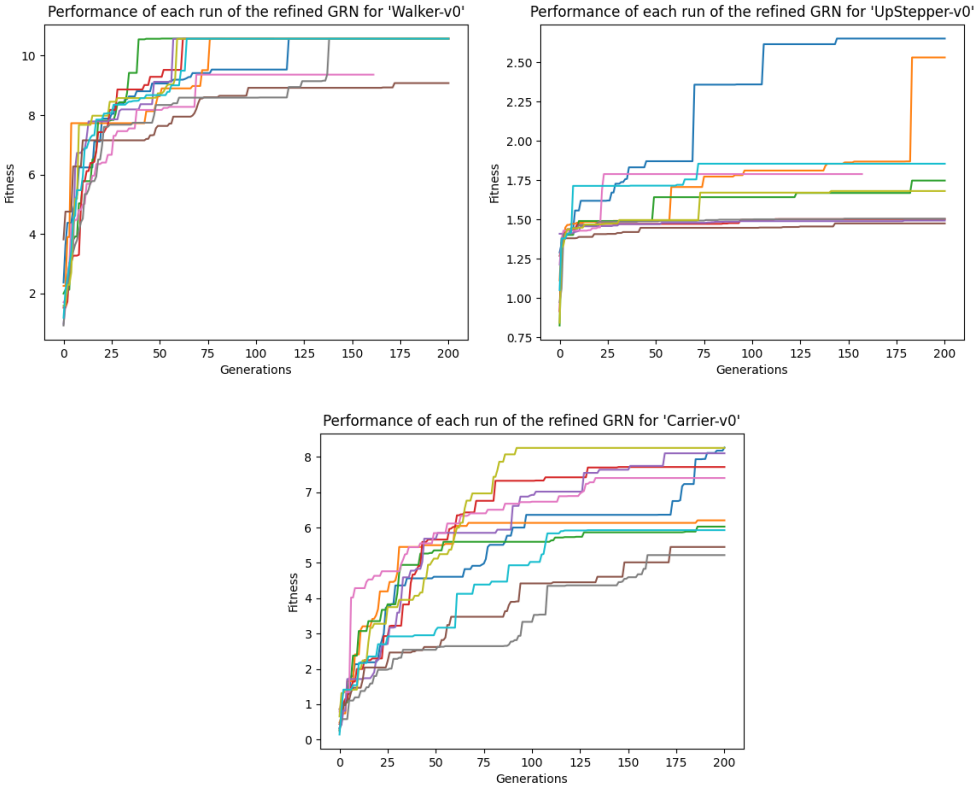
E.1: Initial GRN Performance





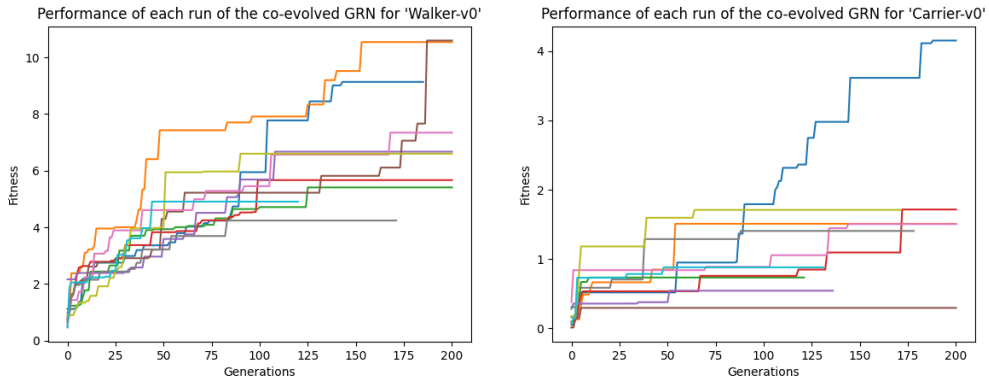
Appendix Fig.1. *Initial GRN fitness for each run*

E.2: Refined GRN Performance



Appendix Fig.2. *Refined GRN fitness for each run*

E.3: Co-evolved GRN Performance

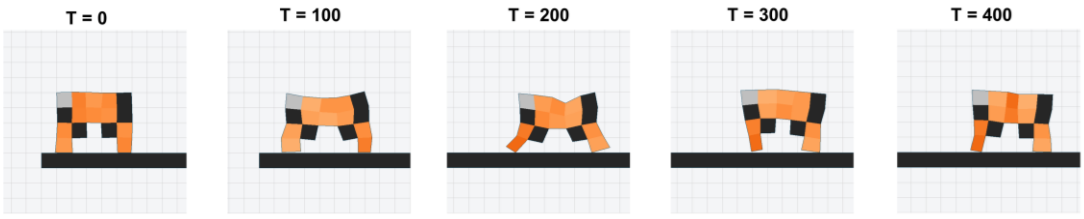


Appendix Fig.3. *Co-evolved GRN fitness for each run*

F. Behaviour and Network Analysis

F.1: Behaviour and Network Analysis for the Initial GRN

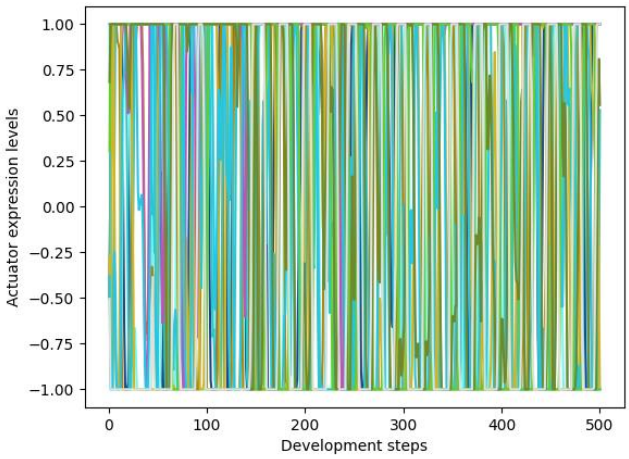
Walker-v0 Task



Appendix Fig.4. *GRN's performance on 'Walker-v0'*

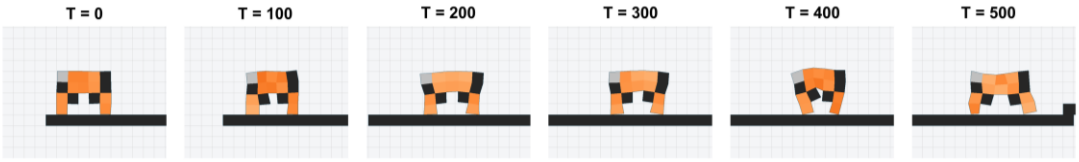
The video reveals that while the robot moves forward, it makes hardly any progress since it repeatedly starts and stops.

The plot of the GRN's actuators reveals that there are no consistent periodic actuation cycles, as the actuators appear to fluctuate randomly:



Appendix Fig.5. *GRN actuator levels for 'Walker-v0'*

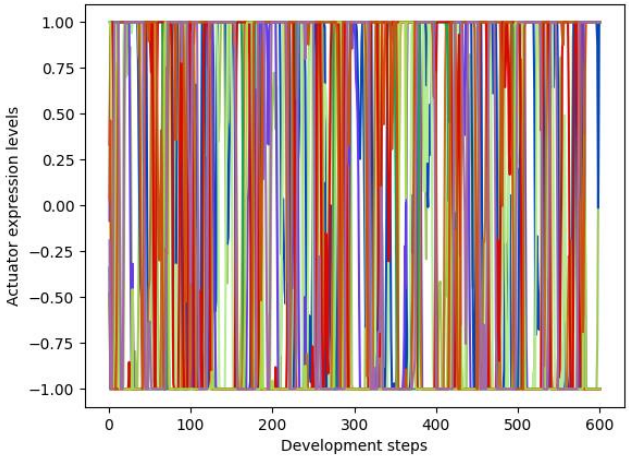
UpStepper-v0 Task



Appendix Fig.6. *GRN's performance on 'UpStepper-v0'*

The video shows that the robot behaves similarly to the Walker-v0 task, where it repeatedly starts and stops. The robot was not able to overcome the first step.

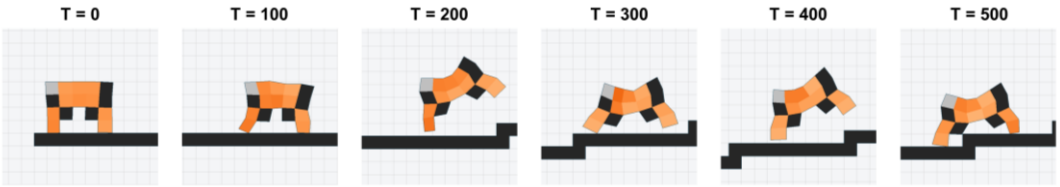
The plot of the GRN's actuators also shows a similar result as previously, where the actuator levels fluctuate with no periodic cycles:



Appendix Fig.7. *GRN actuator levels for 'UpStepper-v0'*

F.2: Behaviour and Network Analysis for the Refined GRN

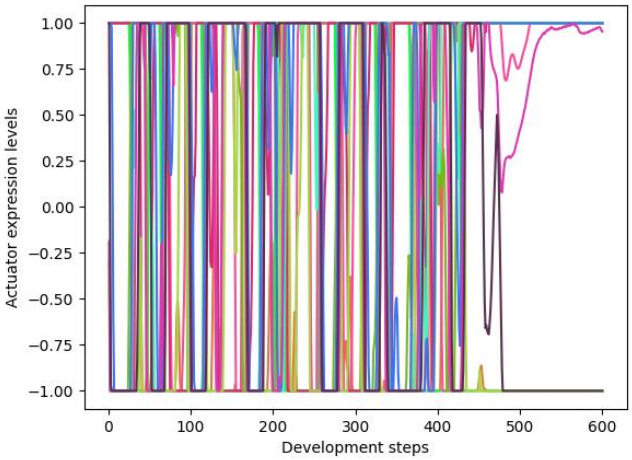
UpStepper-v0 Task



Appendix Fig.8. *Refined GRN's performance on 'UpStepper-v0'*

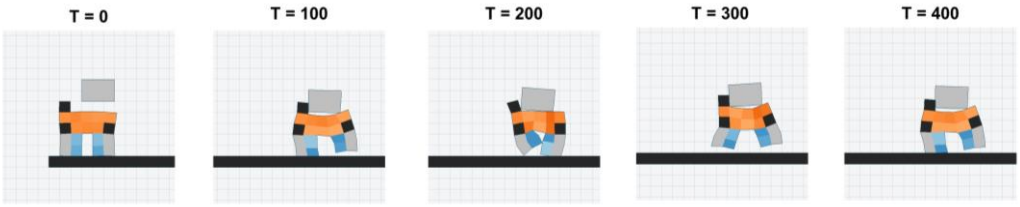
The video shows that the robot initially presents effective actuation which allows it to launch itself over the first step; however, it was not able to progress further since it froze on the second step.

While the plot of the GRN's actuators for this task appears noisy, the behaviour analysis initially indicates effective actuation. However, at around 450 steps, the actuator levels suddenly fall to 1 or -1, which prevents the robot from moving further and causes the robot to freeze in place:



Appendix Fig.9. *Refined GRN actuator levels for 'UpStepper-v0'*

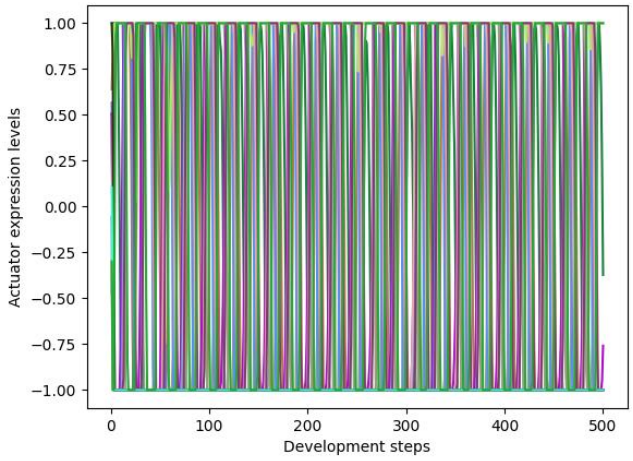
Carrier-v0 Task



Appendix Fig.10. *Refined GRN's performance on 'Carrier-v0'*

The video shows the robot successfully carrying the block while moving; the robot runs fairly quickly without dropping the block. However, it moves slower than in the Walker-v0 task and does not manage to reach the end of the environment.

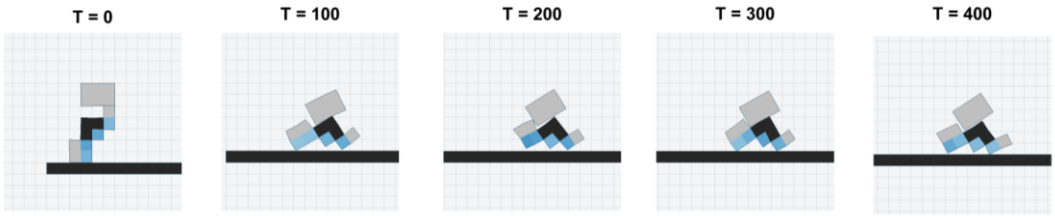
The GRN's actuators for this task quickly fall into a regular cyclical pattern:



Appendix Fig.11. Refined GRN actuator levels for 'Carrier-v0'

F.3: Behaviour and Network Analysis for the Co-evolved GRN

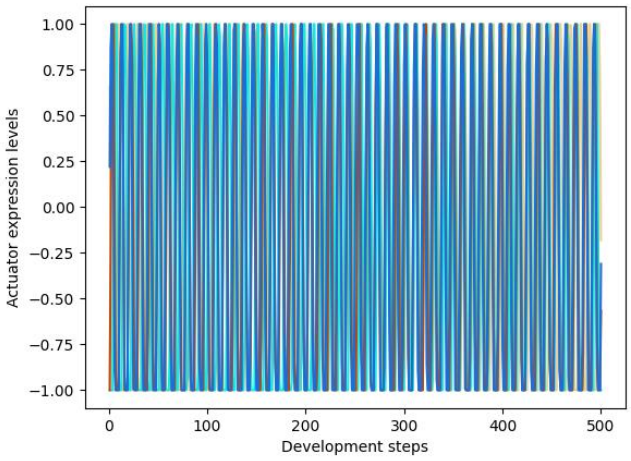
Carrier-v0 Task



Appendix Fig.12. *Co-evolved GRN's performance on 'Carrier-v0'*

The video shows the robot successfully carrying the block along the terrain without dropping it. The robot uses its soft and rigid voxels (shown in grey and black) to prevent the block from falling and uses its actuators (shown in blue) to move forward. As with the previous iteration, the robot moves slower than in the Walker-v0 task.

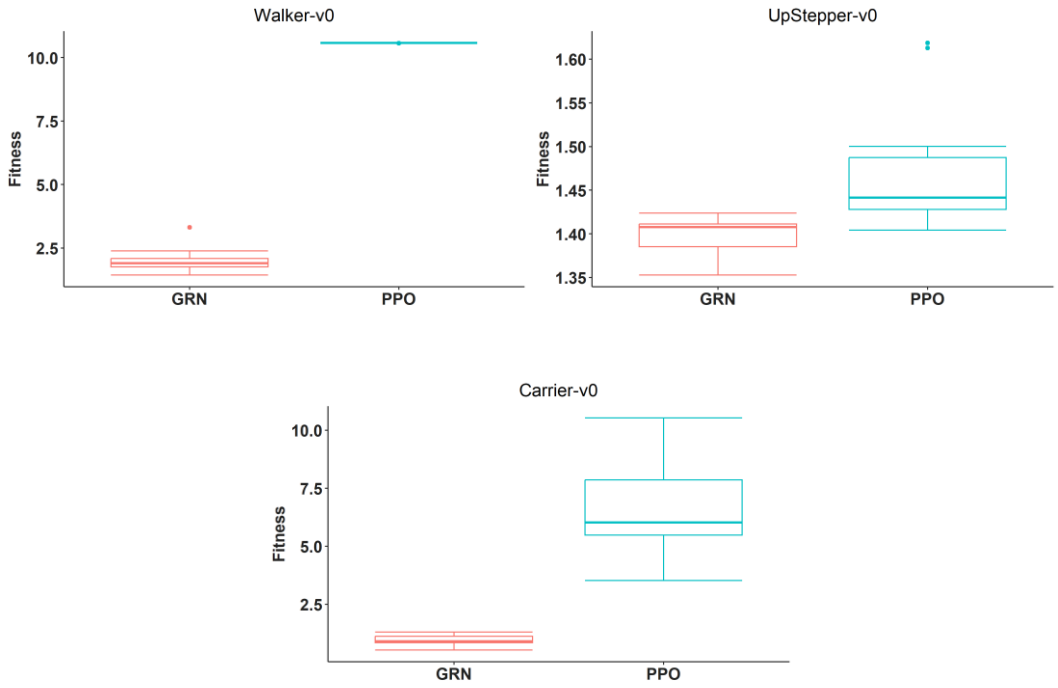
The GRN's actuators fall into a quick cyclical pattern, which allows for effective movements:



Appendix Fig.13. *Co-evolved GRN actuator levels for 'Carrier-v0'*

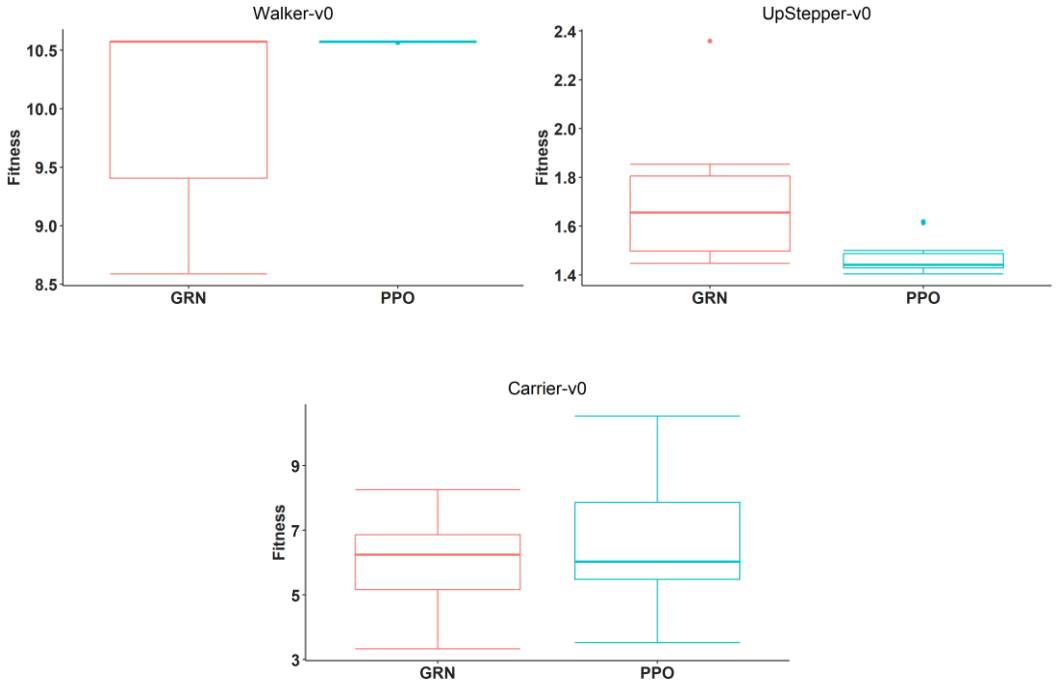
G. Box Plots

G.1: Comparison of initial GRN and PPO performance



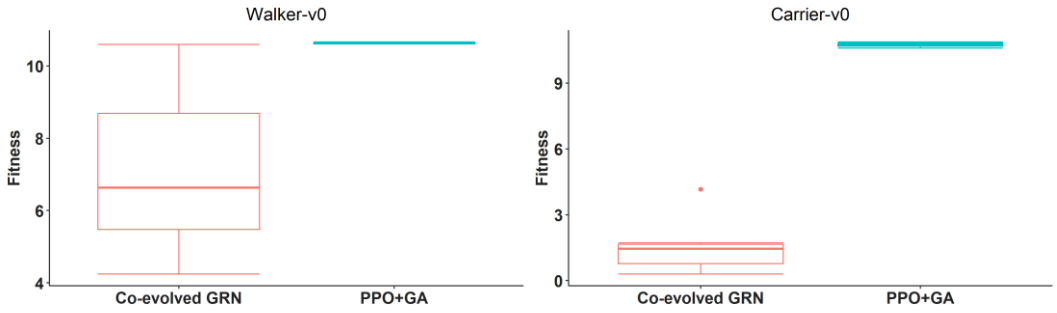
Appendix Fig.14. *Box plots of initial GRN and PPO performance*

G.2: Comparison of refined GRN and PPO performance



Appendix Fig.15. *Box plots of refined GRN and PPO performance*

G.3: Comparison of co-evolved GRN and PPO+GA performance



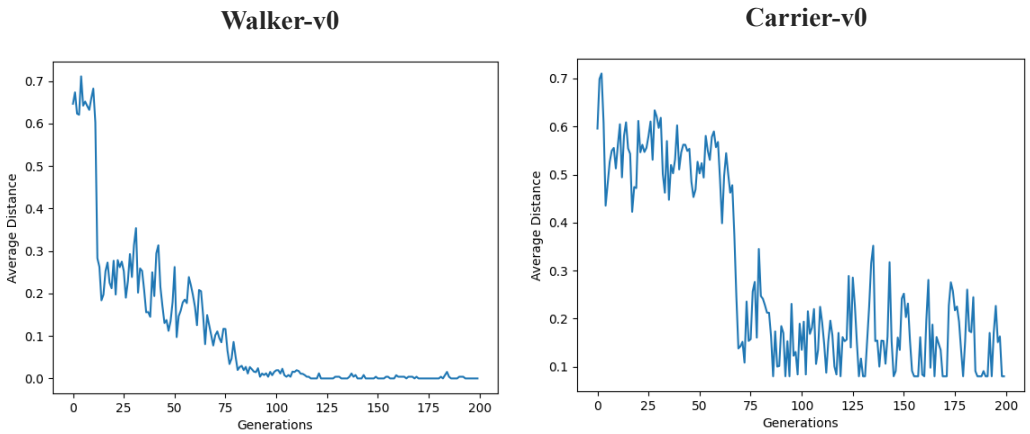
Appendix Fig.16. *Box plots of co-evolved GRN and PPO+GA performance*

H. Diversity Analysis

The results for the co-evolved GRN indicate that the diversity of different robot shapes decreases over time.

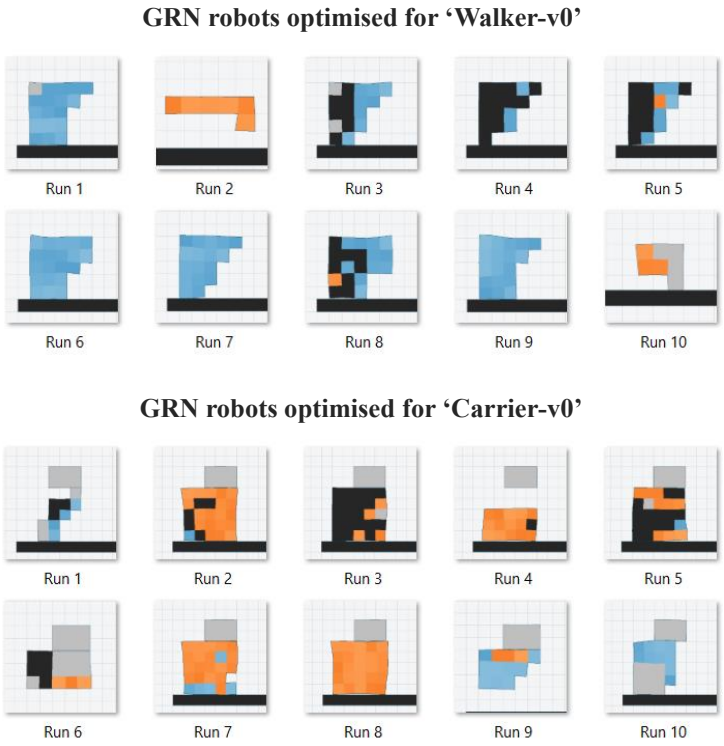
To measure this quantitatively, the average phenotypic (shape) diversity of the top 20 robots in the population has been plotted. This was achieved by calculating the difference in shape similarity between each pair of robots for each generation by counting the proportion of voxels which are the same in both robots.

The results show that the average similarity decreases sharply over generation time for both tasks and remains relatively low:



Appendix Fig.17. *Average shape similarity between the best 20 robots over generations*

I. Best-performing Co-evolved GRN Robots



Appendix Fig.18. *The best-performing co-evolved GRN robot in each experiment*