Cross-Breeding the Cross-Breeders: Improving OpenES with Karl Sims

Christopher Caldas

4th Year Project Report
Artificial Intelligence and Computer Science
School of Informatics
University of Edinburgh

2021

Abstract

The automation of robotic design is a complex task. This is especially so when attempting to optimize the physical components and the software components of a robot simultaneously. This simultaneous optimization is known as Co-Design[1].

A genetic algorithm is an optimization strategy based on the patterns of biological evolution. Genetic algorithms creates generations of potential solutions, and mutate them until the solutions are able to solve the task. OpenAI has created OES,which is an implementation of genetic algorithms, and has found success in solving some optimization tasks. However, it can sometimes struggle with complex tasks due to lack of variety in their mutations.[2].

In order for OES to be more successful with co-design tasks, we have modified OES with features first created in Karl Sim's paper "Evolving Virtual Creatures"[3]. These modifications create more variety in the mutating process, which enables OES to train faster than it normally would on co-design tasks, so long as these tasks are of suitably high complexity. Based on our experimentation on the BipedalWalker task from OpenAI's gym[4], this modification enables OES to train more efficiently on tasks of significant complexity. This could empower OES to be successful in further unsupervised learning tasks of suitably high dimensionality.

Acknowledgements

Thank you Steve Tonnau for the guidance and assistance all year for this project.

Thank you Simon Delmare, Hussein Hamdan, and Tarek Bou Nassif for the computing assistance during a critical time.

Table of Contents

1 Introduction **1**
- 1.1 The Co-Design Method . . . 1
- 1.2 Issues of Co-Design . . . 2
- 1.3 Application of Genetic Algorithms/Evolutionary Strategies . . . 2

2 Background **4**
- 2.1 Introduction to Genetic Algorithms . . . 4
- 2.2 Genetic Algorithm Advantages for Co-Design . . . 5
- 2.3 Introduction to Co-Design . . . 6
- 2.4 Co-evolutionary Robotics . . . 7
- 2.5 Robot Co-design: Beyond the Monotone Case . . . 7
- 2.6 OES . . . 8
- 2.7 Karl Sims . . . 9
- 2.8 OPENAI Gyms and Box2D . . . 11
- 2.9 ESTool . . . 12

3 Methodology **13**
- 3.1 Generic Genetic Algorithm Training Process . . . 13
- 3.2 OES Mutation . . . 13
- 3.3 Karl Sims Methods . . . 14
  - 3.3.1 Cross-Over Method . . . 14
  - 3.3.2 Grafting . . . 14
- 3.4 BipedWalkerPhys . . . 15
- 3.5 Experimental Models . . . 15

4 Results **18**
- 4.1 Biped Walker Easy . . . 18
- 4.2 BipedEasyPhys . . . 19
- 4.3 BipedHardcorePhys . . . 21

5 Discussion **24**
- 5.1 Effects of increasing complexity . . . 24
- 5.2 Hybrid Model Discussion . . . 24
- 5.3 BipedHardCorePhys Discussion . . . 25
- 5.4 Comparison to Reinforcement Learning . . . 27

6 Conclusions **28**

7 Future Work **29**
7.1 Experimentation on Increasingly Complex Tasks in Co-Design 29
7.2 Investigation Into Tasks Outside of Robotics 30

Bibliography **31**

Chapter 1

Introduction

1.1 The Co-Design Method

In present day, robots are being used for more and more tasks both in the commercial and domestic space. It is estimated that the robotics industry will grow from 76.6 billion USD in 2020 to 176.8 billion USD in 2025 due to the ramping up of utilizing robotics in the industrial, manufacturing, and service industries[5]. However, the design of these robots continues to be a challenging and complicated topic. Robotic design currently requires a large amount of expertise, a large amount of time, and therefore a large amount of money. One of the causes for the large expense is due to the common design choice of separating the problems of the robot's physical design and the problem of a robot's software optimization. Solving these problems separately is inefficient[6], due to the fact that these problems are deeply connected.[7]. Separately solving these two problems and then trying to reconnect them becomes a complex process of trial and error. This process does not promise to find an optimal solution, and also requires skilled experts to provide detailed oversight throughout.[8]. For tasks which have military or large-scale industrial application these inefficiencies can be overcome with the tremendous budget that these spaces allow. However with the exponentially increasing number of tasks which people wish to solve via robotics, this is not a sustainable solution. Therefore, more efficient methods of design are necessary.

This motivates the topic of Co-Design. Co-design is the idea of developing both the physical shape and the control methods of a robot simultaneously towards completing a task. The ability to automate robotic design by simply creating a high-level description of the specific tasks which are required for the robot to accomplish would substantially reduce the knowledge, manpower, and expense of robotic development[1]. This paradigm of development has been a topic of recent research and excitement. Having a system of automation which didn't require the fine tuning of previous systems, and was able to tailor the physical design to the task without any skilled human intervention, would be a quantum leap forward in robotic development and research. This would lower the high-level knowledge boundary required for robotic development, allowing for more accessible research.[8]

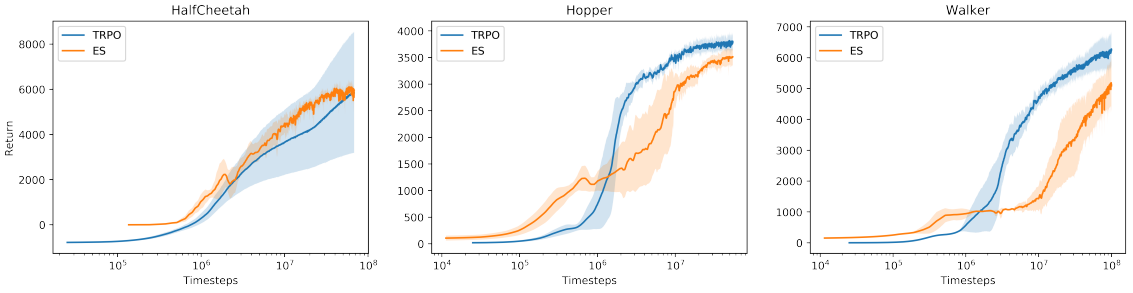
1.2 Issues of Co-Design

As wonderful of an idea as it would be to directly move towards attempting to find the optimal configuration of all components of a robot, this creates a highly dimensional problem. In attempting to solve this problem by a naive search of software optimization can be complicated enough, but the addition of physical characteristics makes an even bigger mess. As is often the issue with high dimensional systems such as the design of a robot, the issue of complexity arises due to all of the components being interconnected. Changing the length of the the limbs may require the motors to be placed differently for optimal gait, which may require the software handling the motor controller to be updated, which may require additional power to be outputted, etc. [9] Not to mention that in many cases with robotics, just calculating the performance of a single configuration can be an expensive operation. Therefore, other non-naive searching methods must be used.

There are many exciting methods which are being researched. In attempting to generate motion, there are two primary methods used. One method is creating a trajectory optimisation problem to solve the task.[10]. The other popular method is with reinforcement learning based techniques. Machine learning, however, has some properties which make it difficult to fully utilize for robotics. Training data for these tasks can be highly limited, and expensive to produce[11].Additionally, in real world scenarios trying to train a neural network by calculating the loss gradients on these can be impossible. Another potential solution to this problem is the application of Genetic Algorithms/Evolutionary Strategies.

1.3 Application of Genetic Algorithms/Evolutionary Strategies

Initially, our research began in comparing using reinforcement learning techniques directly against using genetic algorithms as described by Karl Sims. However, we discovered that extensive work had been done in this field. The team at OpenAI had done an in-depth investigation into the feasibility of using genetic algorithm techniques specifically in comparison to reinforcement learning techniques [2]. In their research, they created a genetic algorithm model called OES. Their results were highly promising, as they found in their experiments that their OES models were competitive with reinforcement learning techniques.



[2]

As seen in the above graph taken their report, OES was able to train to the tasks approx-

imately as quickly as comparable reinforcement learning techniques did. As OpenAI argue in their paper, this shows that genetic algorithms are a viable alternative to reinforcement learning. This alternative would be beneficial especially for co-design, as genetic algorithms hold many advantages over machine learning in this area. These advantages will be discussed in more detail in section 2.1.

However, something of interest was that the evolutionary strategies used by OpenAI, and the strategies described in the Karl Sims paper differ. For one, OES was tested on tasks in which they only were required to train motor controllers, where as in the Karl Sims paper they were being used for optimizing motor controllers in parallel with finding optimal physical characteristics of creatures (aka co-design).[3]. Another difference was in regards to their methods of generating creatures. In the OpenAI methods, the new potential solutions are strictly generated via asexual mutation. . The Karl Sims methods take the ”evolution” part of evolutionary strategies much more literally, in that not only are the most successful creatures mutated by themselves, but they are also (through varies mechanisms which will be discussed in later sections) ”bred” with each other in order to produce more variety in the potential solutions[2].

One of the issues with Genetic Algorithms, as noted by OpenAI, is in regards to the algorithms plateauing for long periods of time, due to lack of variety of outputs. Could this issue be dampened by implementing the methods used in the Karl Sims paper? Additionally, how do the evolutionary strategies perform compared to reinforcement learning techniques when trying to simultaneously train a motor controller and optimize physical characteristics?

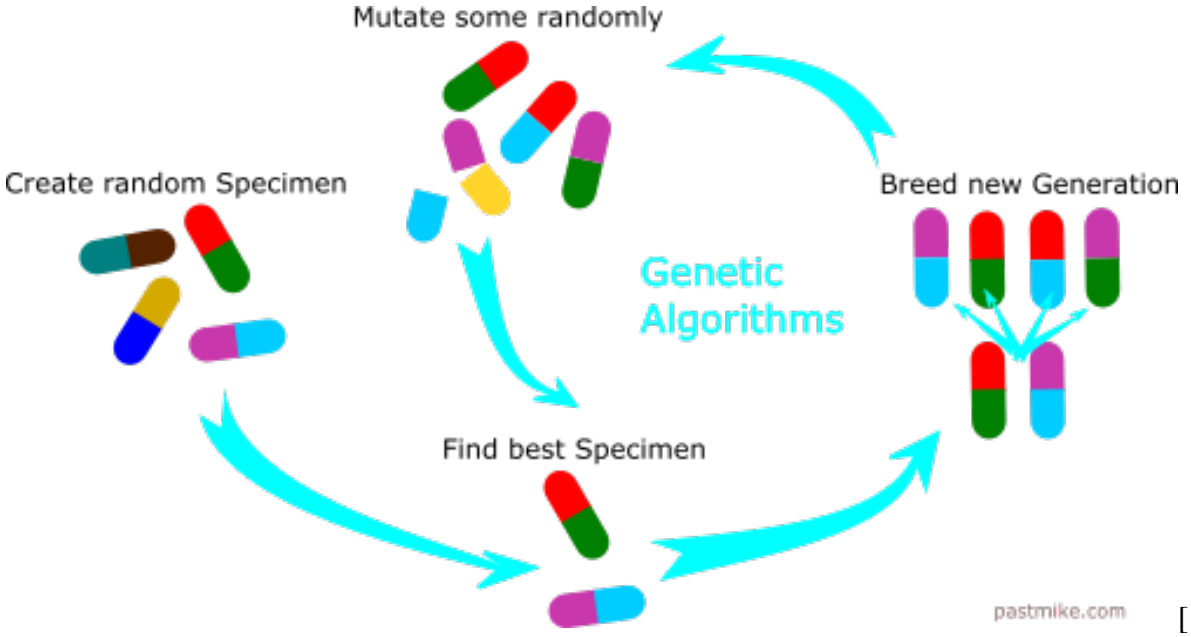
To investigate this question, we created modifications of OpenAI's OES model which, in addition to it's original mutation methods, used Karl Sim's mutation methods. We then trained these models on OpenAI Gym's Bipedal Walker task, and on a modified version of this task which requires the model to generate physical characteristics for the biped. We have found that for increasing complicated tasks, the OES model benefits from the addition of Karl Sim's breeding methods.

Chapter 2

Background

2.1 Introduction to Genetic Algorithms

Genetic algorithms as a concept are nothing new. In fact, discussion regarding using evolutionary strategies for computing simulation was theorized in the 1950s by Alan Turing[12], and was even being tested as early as the 1960s by Nils Aall Barricelli[13]. While there has been a great deal of innovation and experimentation since genetic algorithms were first proposed, the general concept has remained the same.



[14]

Genetic Algorithm Steps

- Initialize

  For whatever task it is you are trying to solve, you first generate a population of potential solutions. There are a few ways which this can been done, including guided random generation, and seeding the population from known successful solutions, but so long as the population will produce valid attempts at the task

any method is fine.

- Fitness Score

  Each member of the population must then be scored by a ”fitness function”, giving a score to how successful this solution is. For a simple example such as attempting to fit a function to a line, this could be calculated by distance between the line and the function, however for more complicated tasks the fitness function can be difficult to define. If a task is able to have relatively simple fitness function defined for it, it is much more likely to be able to be solved by genetic algorithms. Otherwise, this can be a point of weakness to this strategy. Once the population has been scored, the population will be ranked based on their fitness score. The top ranked population (by some predefined survival ratio) will ”survive”, while the other solutions will be forgotten.

- Cross-Breeding and Mutation

  These surviving solutions will then be cross-bred and mutated in order to produce the next generation of potential solutions. This is where the largest variation in between the strategies exist. Cross-Breeding is the process of taking two of the solutions and combining portions of the two, into a new single solution. Mutation is the process of taking one of the solutions, and randomly altering it's sections. An important point to note is that after the cross-breeding and mutating is finished on the surviving solutions, it is necessary for the population of this new generation to be the same size as the initial population.[15]

- Finishing

  This process is repeated over and over, until either no further significant improvements on the survival scores are found,a given number of generations have occurred ,or if the fitness score is sufficiently high enough for the purposes of the task (the task is solved).[15]

This process is highly modifiable and there have been several variations, two of which will be explored in depth in this paper (Karl Sims and OES).

2.2 Genetic Algorithm Advantages for Co-Design

In the co-design space, or more generally in the un-supervised learning space as a whole, genetic algorithms hold several advantages over reinforcement learning techniques. All of these advantages are what motivate further research into genetic algorithms, as having a reliable alternative to reinforcement learning would further empower research. To summarize, here are several of the benefits.

Genetic Algorithm Benefits for Non-Supervised Learning

- No Back Propagation

  One large advantage is in regards to the fact that it is unnecessary to calculate an error gradient by solving for the back-propagation of errors. In real world applications such as robotics in which there are exponentially many variables to

account for, which can make back-propagation to calculate the gradient highly complicated. The ability of genetic algorithms to simplify everything down to the fitness score is a tremendous strength.[2]

- Parallel Training

  Another big perk of genetic algorithms is that it is perfectly suited to using parallel computing in the training process. By the nature of Genetic Algorithms having a population of potential solutions which all need to be evaluated without the outcomes of this evaluation impacting the others, this allows for all the solutions to be trained simultaneously. In comparison to other optimization techniques in which there is one solution which is modified during each update, genetic algorithms are able to be trained much more efficiently.[2]

- Higher Robustness

  Genetic algorithms require far fewer hyper-parameters to be set in order to train a model. This means less trial and error testing of which hyper-parameters will cause a model to fail to train a task.[2] Additionally, genetic algorithms utilize probabilistic transition rules in comparison to traditional methods which utilize deterministic methods[16]. Both of these points lead to genetic algorithms being easier to apply to a wider range of tasks.

- Better suited for long episode times

  In tasks where the ”episode” takes a longer time (in other words, that the fitness score takes more time to calculate for each creature), ES becomes a more attractive choice. This is seen in OpenAI’s work from comparing their respective gradient estimators.[2]

2.3 Introduction to Co-Design

The topic of co-design in robotics is an exciting one, as the ability to efficiently use co-design on real world, ”messy” applications would be a tremendous break-through in robotics, and allow of a flourishing of innovation. However, with traditional automated design methods, this has not been possible.

To reiterate, co-design in robotics is explained by the following. Taking a given task, and then simultaneously optimizing the physical components (motor placements, limb lengths, etc.) and the software aka ”mental” components (motor controllers, power output, etc. While it may seem trivial that this would be the best way to go about creating a design, in real-world applications this has not been feasible with traditional methods.

Even for tasks that seem simple, attempting to optimize both the physical and mental components of a design simultaneously exponentially complicates the problem. While it has been feasible for simplified simulated models with limited parameters, it quickly becomes too expensive to calculate with traditional methods as the task becomes more complicated.[9] This historically has meant that accomplishing this requires software

to create simplified models, followed by substantial hyper-parameter tuning and highly skilled work from experts in the field[1].

2.4 Co-evolutionary Robotics

Co-design as a design principle has gone by various terms. In 1999, a paper called ”Co-evolutionary Robotics” did work into what we would now call co-design, but instead referred to the task as fully automated design.[17]

Their experiments are very much in line with ours, as they attempt to use genetic algorithms to train a neural network in order to accomplish co-design. However, their experiments had several limitations due to their current state of the art. Regardless, they were able to show successful and robust solutions on simple tasks inside CAD simulation. They do note that these CAD models would not be able to be built in the real-world due to lack of ability to have meaningful constraints in their CAD software.

An interesting historical note regarding this paper is that they mention that fully automated design is so difficult due to the nature of engineering at the time. To quote the paper directly, ”The automatic design idea is perhaps the most challenging, as it entails imitation of one of humanity’s most prominent acts of intelligence:creativity[17].”

As this paper was written before recent major advances in Machine Learning and Artificial Intelligence, part of their writing is in regards to the fact that the engineering standards of the time still mostly relied on human expert intervention, and therefore the computer simulations were not required (and not able) to generate their own unique designs. It is exciting to consider how thrilled the researchers writing this paper would be to know exactly how far computers have come in the aid of robotic simulation and development.

2.5 Robot Co-design: Beyond the Monotone Case

In this paper, motivated by the technological improvements in computing furthering the research in robotic design, they looked into ways to optimize co-design.The method they investigated for this purpose was a binary optimization. [8]

Binary optimization in and of itself is simply trying to find the best combination of attributes to optimize some function. To tailor it for the purposes of co-design, they had to designate all the various attributes of the design of the robot into binary parameters. This includes, to quote their paper, ”system-level performance, system-level constraints, and intrinsic constraints”.

Additionally, as they discuss, binary optimization problems are easily solvable by computers as long as some properties hold (such as, as they mention, linearity).

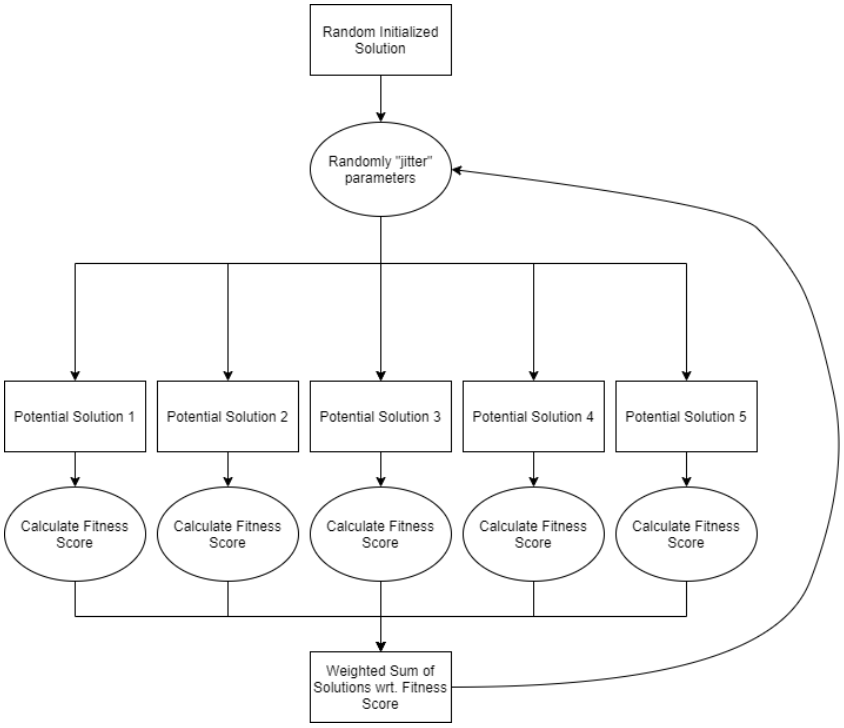
They investigated this method on autonomous drone racing and co-design of a multi-robot system. They found that their binary optimization techniques were highly successful and efficient in solving these tasks.

2.6 OES

OES is OpenAI's approach to using evolutionary strategies (another term for genetic algorithms in this context). We have already explained the general pattern of genetic algorithms, so in this section we will focus on the specifics which differentiate their strategy from the norm.

In their strategy, they begin by randomly initiating a potential solution in the form of a number of parameters, set up as a function such that all the parameters go in, and 1 number comes out in the form of the fitness score. They generate several different variations of this singular potential solution by "jittering" it's parameters. They then evaluate all of these variations on the task, and then create a new potential solution from the weighted average of these variations. The weight being in respect to the associated fitness scores.[2]

To clarify, here is an illustration of this process.



Their lack of cross-breeding could be the cause of one of the weaknesses of ES which OpenAI note in their paper, which is that, "in order for ES to work, adding noise in parameters must lead to different outcomes to obtain some gradient signal". In other words, it can be difficult with genetic algorithms to produce different enough potential solutions from the essentially random generation which it uses. This issue of not having enough variety can be alleviated by applying techniques used by Karl Sims in his paper Evolving Virtual Creatures.

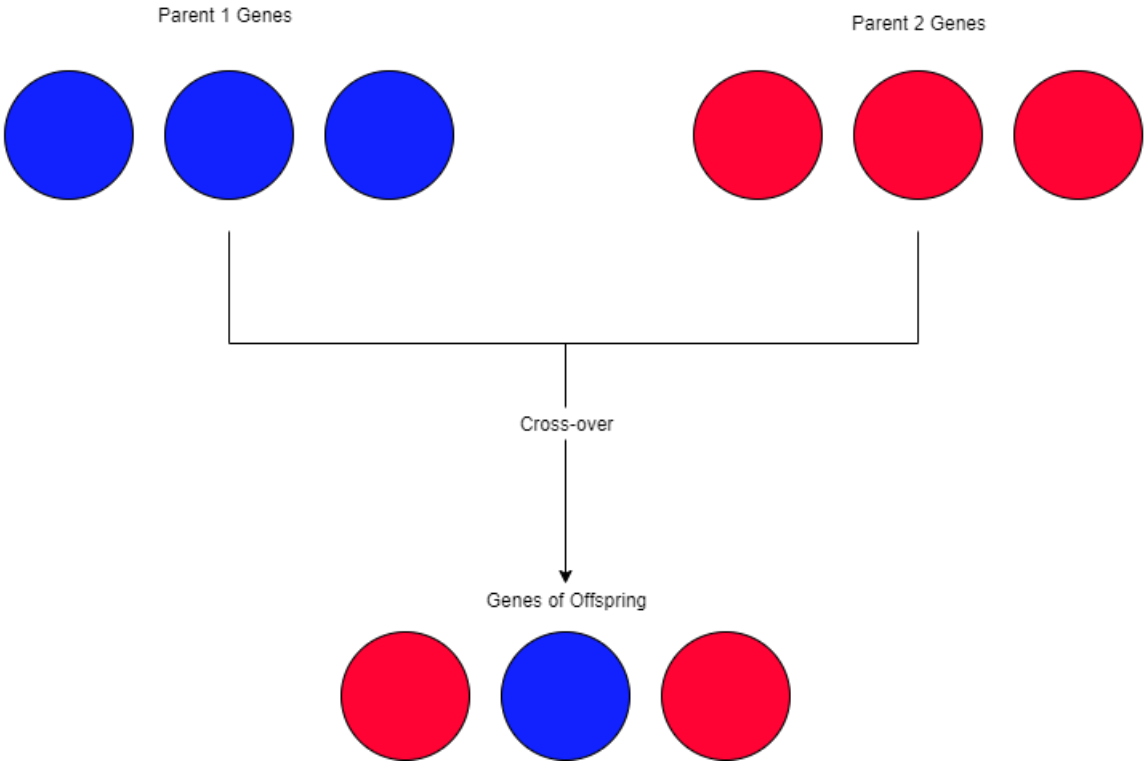
2.7 Karl Sims

Karl Sims, inspired by previous works regarding genetic algorithms, researched into applications into the co-design problem[3]. He wanted to investigate the ability of genetic algorithms to simultaneously optimize the physical design and the motor control for robots for various tasks such as running, jumping, and swimming.

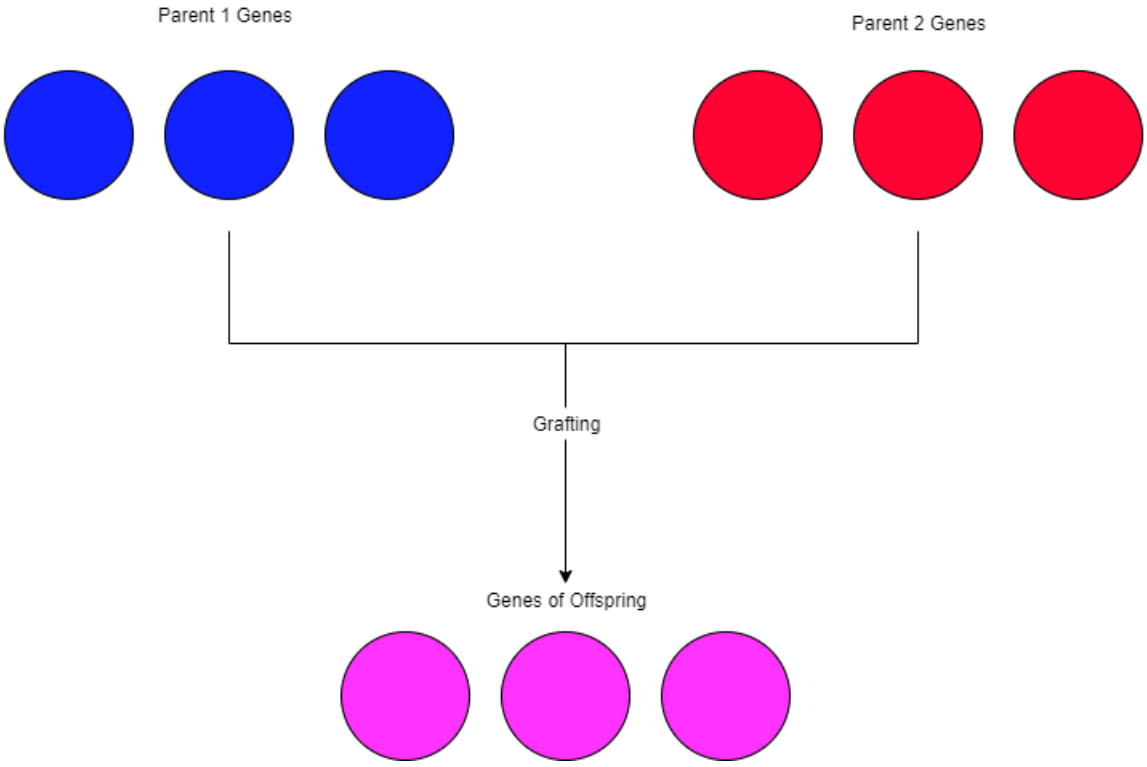
Additionally, this paper proposed the idea of making the mechanisms for mutating new generations much more similar to the biological process of mating. In this paper they created artificial genes for each of the potential solutions, as a means of storing the data related to the morphology of the creatures. These genes were used for the other novelty of this paper which is implementing sexual reproduction patterned off of the real-world biological process.

This paper uses three different methods of creature generation. The primary method is asexual, which is the method most genetic algorithm based strategies utilize. In this method, each creature is bred one at a time. For each gene, there is a chance that the values held in the gene will be shifted around by a small percentage. This allows the creatures who had the higher fitness scores to maintain most of their form which allowed them to produce a high enough fitness score to survive, while also giving them a slight chance to develop new behaviors which will allow the creature to perform better in the next generation. However, as is the case with real life asexual vs sexual reproduction, this method of breeding is much less prone to extreme mutations which could cause substantial increases or decreases in the fitness scores. For this reason in the Karl Sims paper, this method is used for 70 percent of the reproduction, to provide a safe base for the generations.

The other two methods used are both sexual methods, cross-over and grafting. As sexual methods, doing cross-over and grafting requires selecting two virtual creatures to merge their genes. Beyond that, the methods are relatively similar. In cross-over breeding, each gene from the child creature is directly taken by one of the parents which was randomly selected. For instance, if the genes of parent A were (a,b,c), and the genes of parent B were (d,e,f), than a possible result of cross-over breeding of the two could be (a,e,f) or (d,b,c). This breeding method is illustrated in the following diagram



Alternatively, in grafting, each gene taken is a combination of both the parents. So in this instance if the genes of parent A were (a,b,c) and the genes of parent B were (d,e,f), then the genes of the child would be (a/2 + d/2,b/2 + e/2,c/2 + f/2). This breeding method is illustrated in the following diagram

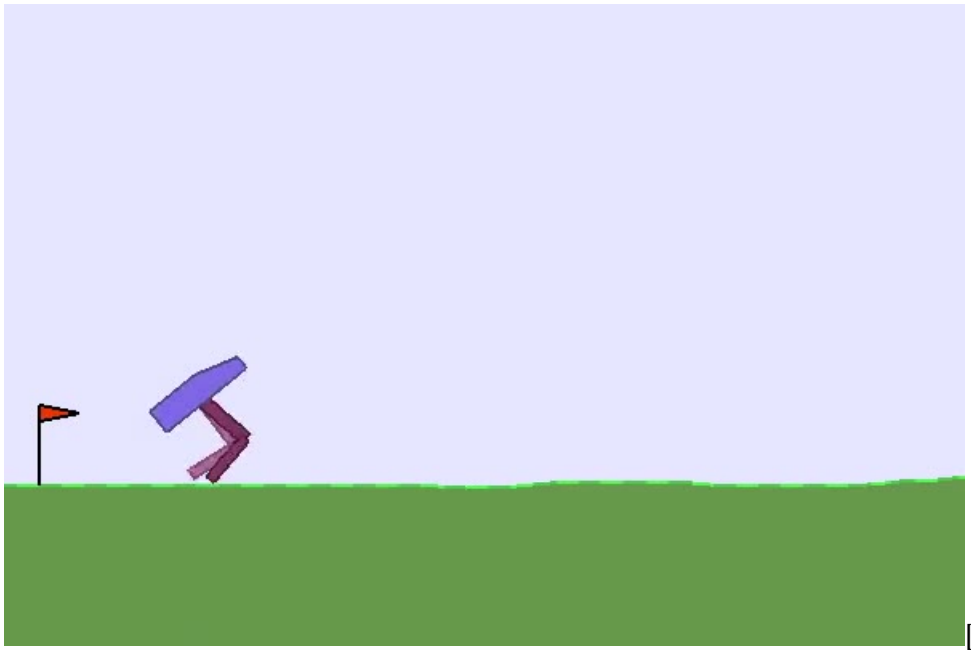


These methods produce a much greater change in the new creatures than asexual methods do. A common drawback of genetic methods is the potential for long plateaus as the algorithm waits for a needed mutation to occur. By having the more high variance breeding methods, this empowers the algorithm to potentially break through those plateaus faster.

2.8 OPENAI Gyms and Box2D

OpenAI's gym project was created as an easy framework for researchers to use to test and experiment on their reinforcement learning models[4]. Additionally, the gym has some built in tasks to test models on. This makes it easy to try a model on tasks of various types. This ease of accessibility additionally has the benefit that many models have been trained on these tasks, which gives the researcher a great deal of baselines for comparison.

OpenAI have created a variety of tasks for testing models. For our case, we will be using the Bipedal Walker task. This task was created using Box2D, which is a simple to use 2D engine. [18]

 [4]

In this task, a bipedal robot is dropped into a randomly created terrain. The task provides the model with 10 LIDAR measurements from the enforcement per step, and the model is required to output 4 numbers which control the 4 motors of the biped. Additionally, the task handles the calculation of the fitness score, by returning a score related to a combination of the distance covered by the biped, and the amount of energy used by the motors.

For the purposes of our experiments some modifications had to be made to the Bipedal Walker task. In the original task, the model only had to generate the actions for the motor controllers. In our updated task, the model has to both control the motor controllers, and provide some physical characteristics of the model(length and width of the

limbs). This change had to be made in order to investigate the various models ability to optimize co-design.

2.9 ESTool

ESTool is a project created for the purposes of comparing and contrasting various genetic algorithms (aka Evolutionary Strategies) on OpenAI Gym tasks[19]. It provides the architecture necessary for training generations in parallel, and stores the results in an easily accessible and comparable way.

The project contains several genetic algorithm strategies built into it, including the previously discussed OES model. By modifying this pre-packaged strategy and adding methods patterned after the Karl Sims paper, we were able to compare the strategies.

Chapter 3

Methodology

For testing our improvements on the OpenAI Evolutionary Strategies, we used the previously mentioned ESTOOL with the addition of our strategies.

3.1 Generic Genetic Algorithm Training Process

Previously we showed a diagram, and described the concepts of genetic algorithms. Now we will describe them as a function.

Result: Trains a model for a given task
Let X = an array of randomly generated models which provide valid solutions to the task;
Let Y = an array the same size as X **while** *Training is continuing* **do**
    for *i = 1 to size(X)* **do**
        Y[i] = fitnessScore(X[i]);
    end
    X = mutate(X,Y);
end

Algorithm 1: Generic Genetic Algorithm

Clearly, the big abstraction in this description is the magical mutate function. Largely, genetic algorithm's differ in how their mutate function operates. The common thread between all mutate functions however is that they all will only operate on the highest scoring solutions from X based on their particular survival ratio, and they will all maintain that the size of X will be the same at any given iteration of the loop.

3.2 OES Mutation

OES handles mutation exclusively by what Karl Sims would refer to as asexual methods. In Section 3.2 of this report we have an illustration depicting this process. OES mutation functions via the following steps.

- Randomly generate a potential solution

- Create a number of mutations of this single solution via randomly shifting all of it's parameters
- Score all of these solutions via a fitness score
- Create a new potential solution out of the weighted average of the mutations, weighted in relation to their fitness scores

[2]

3.3 Karl Sims Methods

The Karl Sims mutation functions also partially include the single-agent "jittering of parameters" mutation which OES uses, but also introduces cross-mating mutations for increased variance. There are two methods of this, Cross-Over and Grafting. In the experiments of the paper, they found the most success with a model that used 70 percent asexual mutation, 15 percent Cross-Over mutation, and 15 percent Grafting mating

3.3.1 Cross-Over Method

In this method, half of the parameters are taken from one model, and half the parameters are taken from the other. The diagram in section 2.5 explains the concepts. Here we have Cross-Over as described via code.

```
def crossover(self):
  output = np.zeros([self.quar_popsize, self.num_params])

  #a and b are both randomly selected amongst
  #the surviving potential solutions from the
  #previous generation.
  a = self.bestOnes[random.randrange(self.quar_popsize)]
  b = self.bestOnes[random.randrange(self.quar_popsize)]
  for i in range(self.quar_popsize):
  #mask1 and mask2 are applied to a and b, setting
  #half of a's value's to 0, and the other matching
  #half of b's values to 0 as well.
    mask1 = np.random.binomial(1, .50, size=self.num_params)
    mask2 = (mask1 - np.ones(self.num_params)) * -1
    output[i] = (a * mask1) + (b * mask2)
  return output
```

3.3.2 Grafting

In this method, each parameter of the new model is created as a combination between the two parent models. Again the diagram in 2.5 better illustrates the concepts, but here is the code explanation.

```
    #a and b are both randomly selected amongst the
    #surviving potential solutions from the previous #generation
  def grafting(self):
    output = np.zeros([self.quar_popsize, self.num_params])
    a = self.bestOnes[random.randrange(self.quar_popsize)]
    b = self.bestOnes[random.randrange(self.quar_popsize)]

    #mask1 and mask2 are applied to a and b, scaling a's values
    #by a randomly generated number,which we can call x, and
    #scaling b's values by 1-x
    for i in range(self.quar_popsize):
      mask1 = np.random.rand(self.num_params)
      mask2 = np.ones(self.num_params) - mask1
      output[i] = (a * mask1) + (b * mask2)
    return output
```

3.4 BipedWalkerPhys

In the original BipedalWalker task from OpenAI Gym, all that is required to solve the problem is to give the biped values to control the motor controller[4]. In order to be investigating co-design, it is necessary for our models to be attempting to optimize both the motor controllers, and the physical characteristics of the biped simultaneously.

To this point, we have created a new task BipedWalkerPhy, which is a modified version of BipedWalker-v2 with the addition of the task needing to be fed 3 additional values which modify the limb lengths of the biped. Unlike the motor controllers, these values can not be changed per each step of a simulation, and can only be set at the start of an episode.

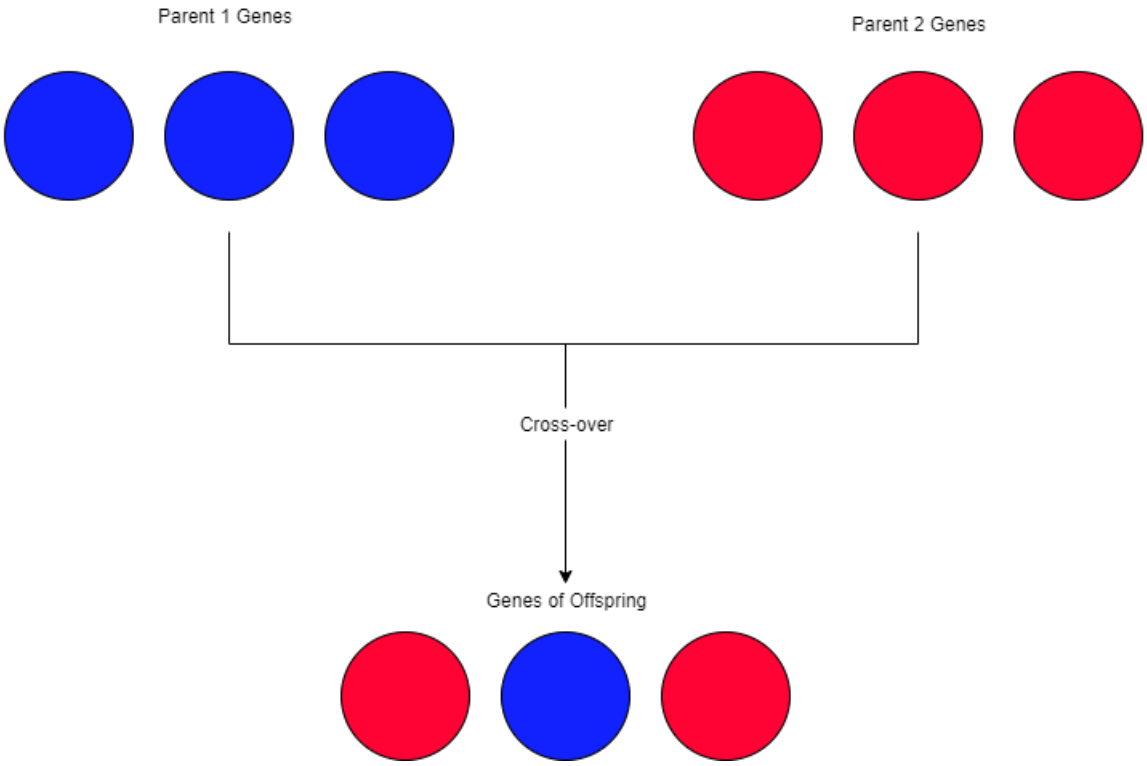
The original BipedalWalker task has two different versions of different difficulty, bipedeasy and bipedhardcore. The difficulty level between these two games is quite large Bipedeasy takes approximately 8 hours to solve, and bipedhardcore can potentially take weeks. The difference comes from the terrain generation, in that the bipedhardcore randomly generates terrain which is difficult for a biped to traverse. BipedWalkerPhys also has these two difficulty settings.

3.5 Experimental Models

In order to gain the benefit from the Karl Sims models, some modification of the OES model must be done. For one, in the original OES method only 1 potential solution is saved after the fitness scores are calculated. For the purposes of cross-breeding, we need to save to memory the best performers of each generation for one iteration. This has the downside of making the algorithm less memory efficient, but we believe the trade-off in performance makes it worth this downside.

Following this, for the purposes of our experimental models the number of mutations

generated to be tested are reduced by 25 percent. This 25 percent is instead generated by the various Karl Sims strategies, executed on the saved previous generation. This process is illustrated in the following diagram.

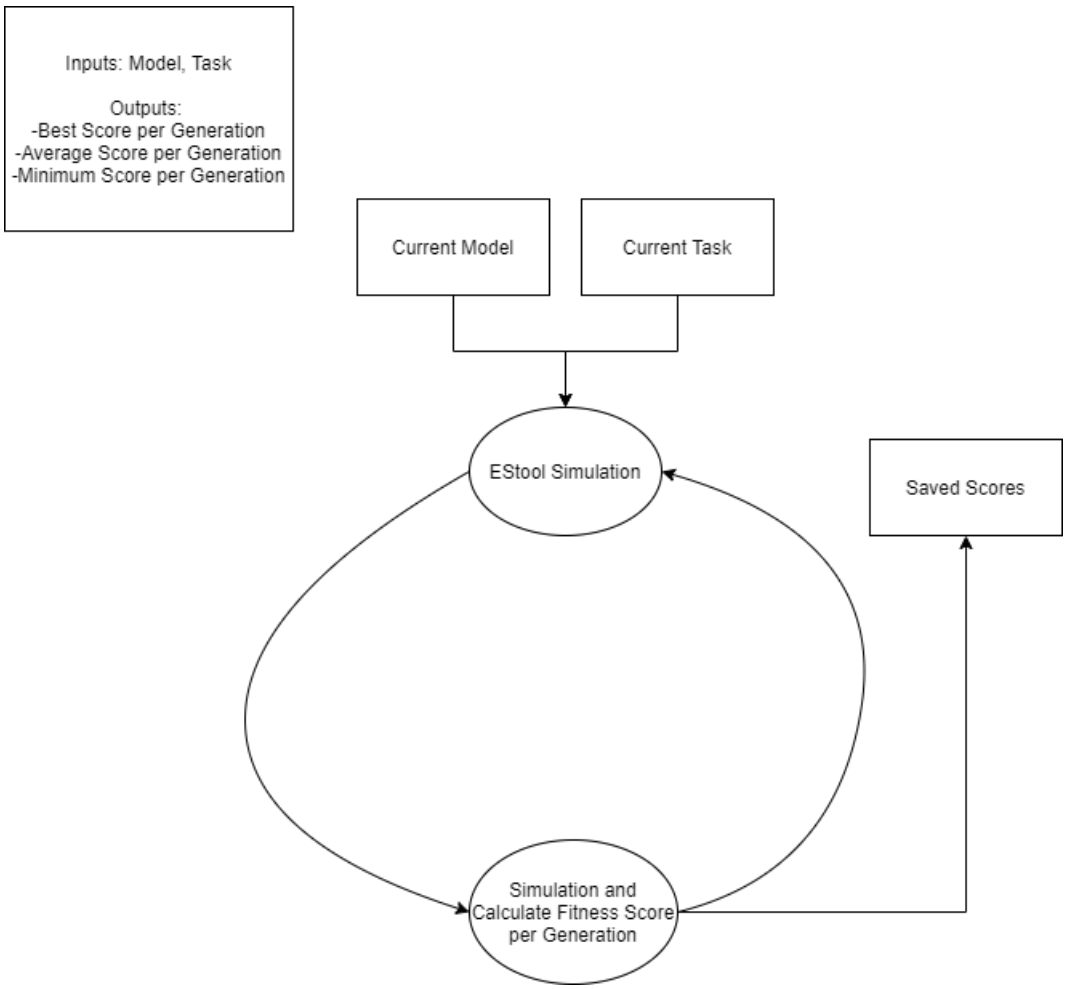


With this design, we have 4 models to test. The standard OES, OES using Karl Sims' Cross-over method, OES using Karl Sim's Grafting method, and OES using both of Karl Sims' methods, which we will from now on refer to as "hybrid".

Using ESTool, we will be testing our 4 different models on both the BipedalWalker and BipedalWalkerPhys gym tasks, keeping all command line variables the same to ensure consistency in the experiments. For our experiments, our ESTool settings were as follows.

- MPI Processes 4
- Number of Workers per Process 4
- Random runs per Model 16

The workflow of our experiments is illustrated in the following diagram.

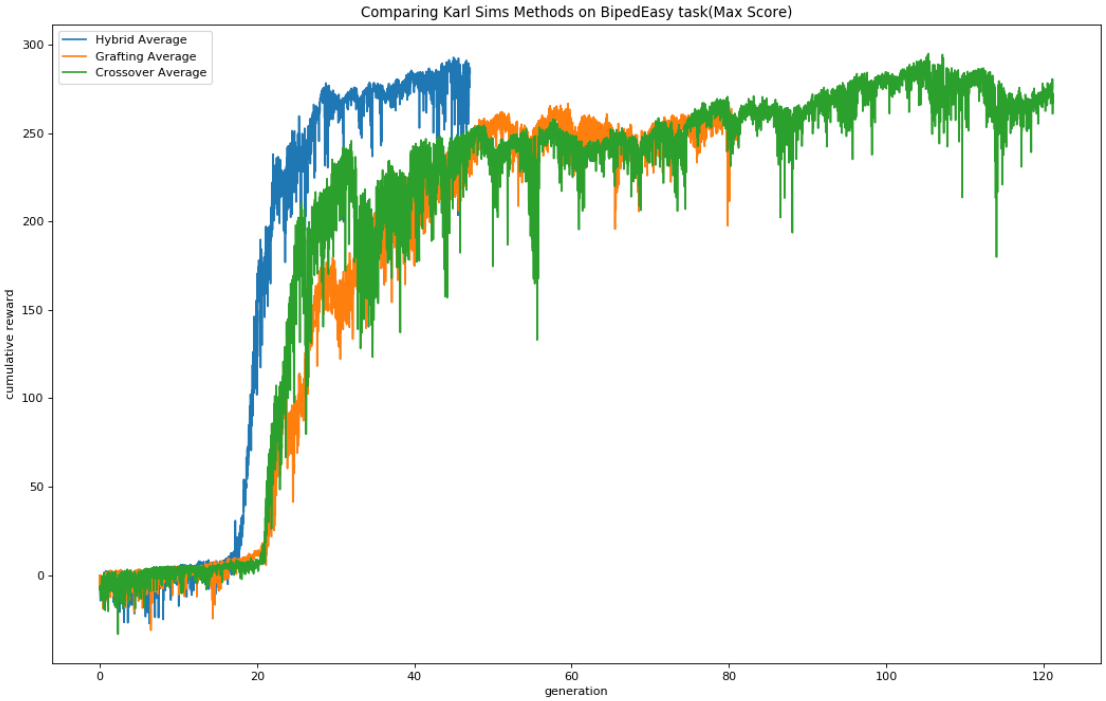

Inputs: Model, Task
Outputs:
-Best Score per Generation
-Average Score per Generation
-Minimum Score per Generation
Current Model
Current Task
EStool Simulation
Saved Scores
Simulation and
Calculate Fitness Score
per Generation

Chapter 4

Results

4.1 Biped Walker Easy

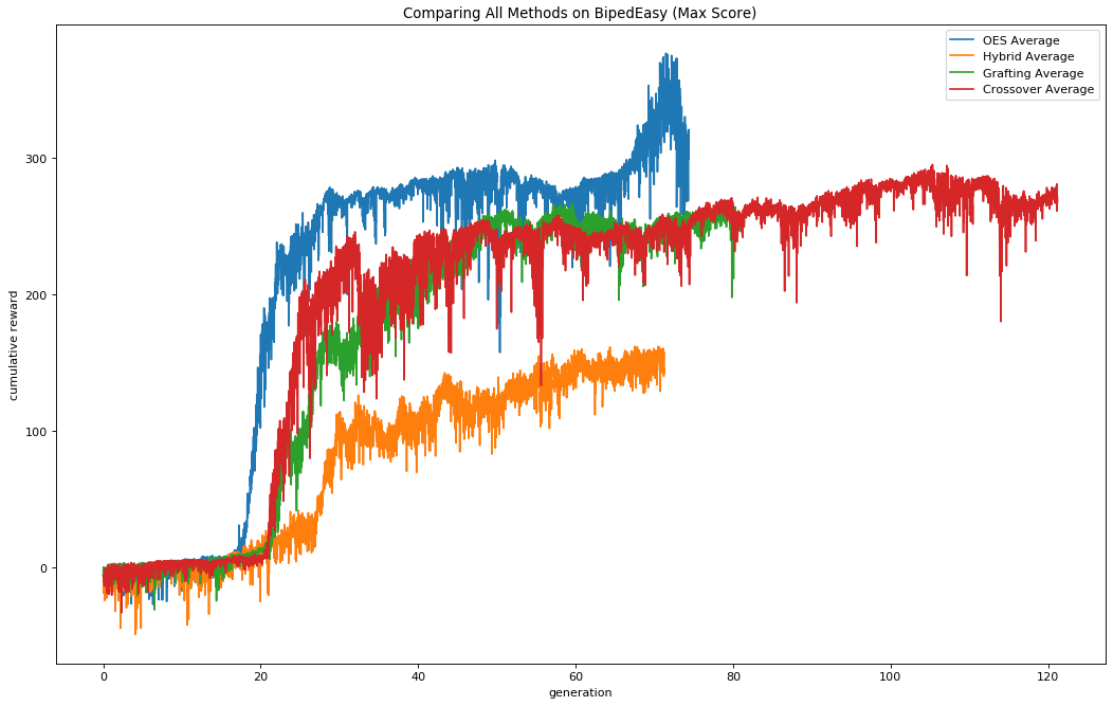
Experimentation began with attempting to solve the BipedEasy task with 3 variations of the Karl Sims, to both ensure the models are functioning correctly, and investigate how well the two Karl Sims methods are fit to this task.



As expected based on Karl Sims work[3], the model which was the fastest to train on the task was the hybrid model.

Continuing on, we then compared the performance of the Karl Sims models, with the

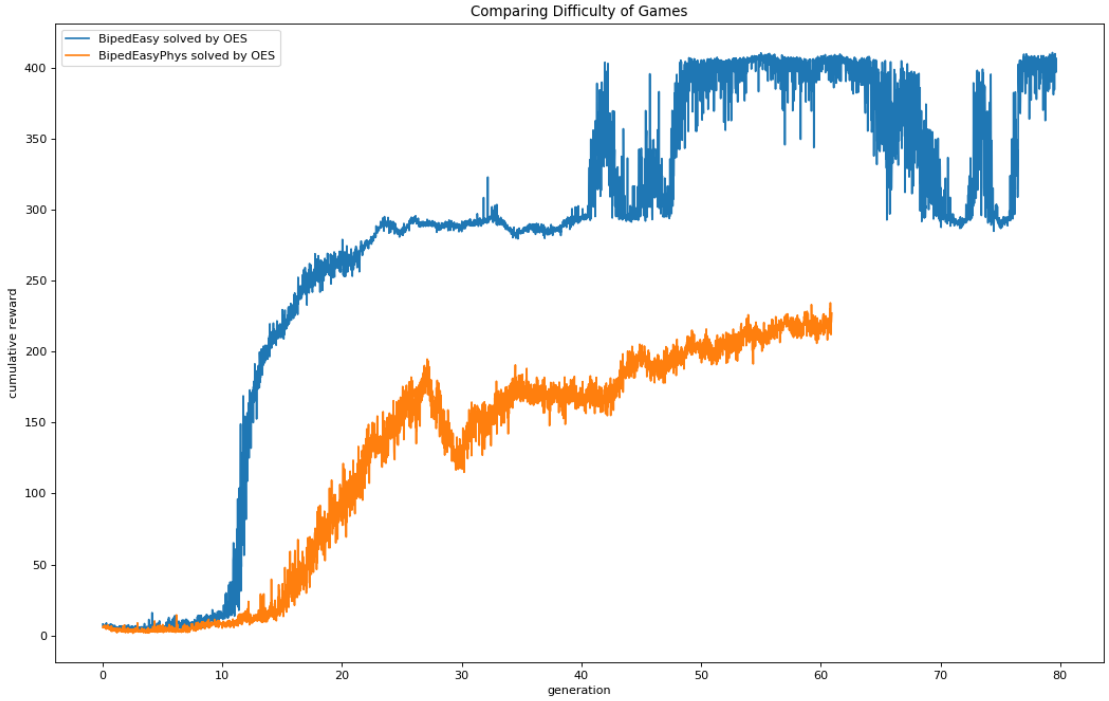
original OES model.



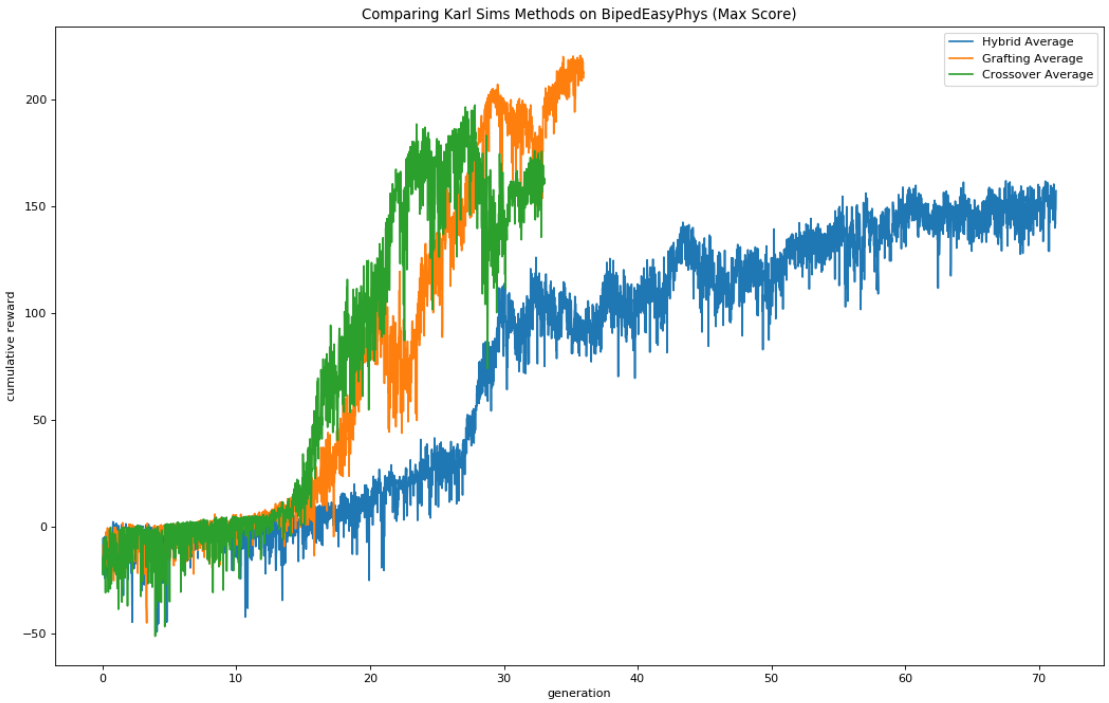
For this task, while the Karl Sims models were able to perform similarly to the OES model, the OES model was able to generate a successful solution faster than the Karl Sims Models.

4.2 BipedEasyPhys

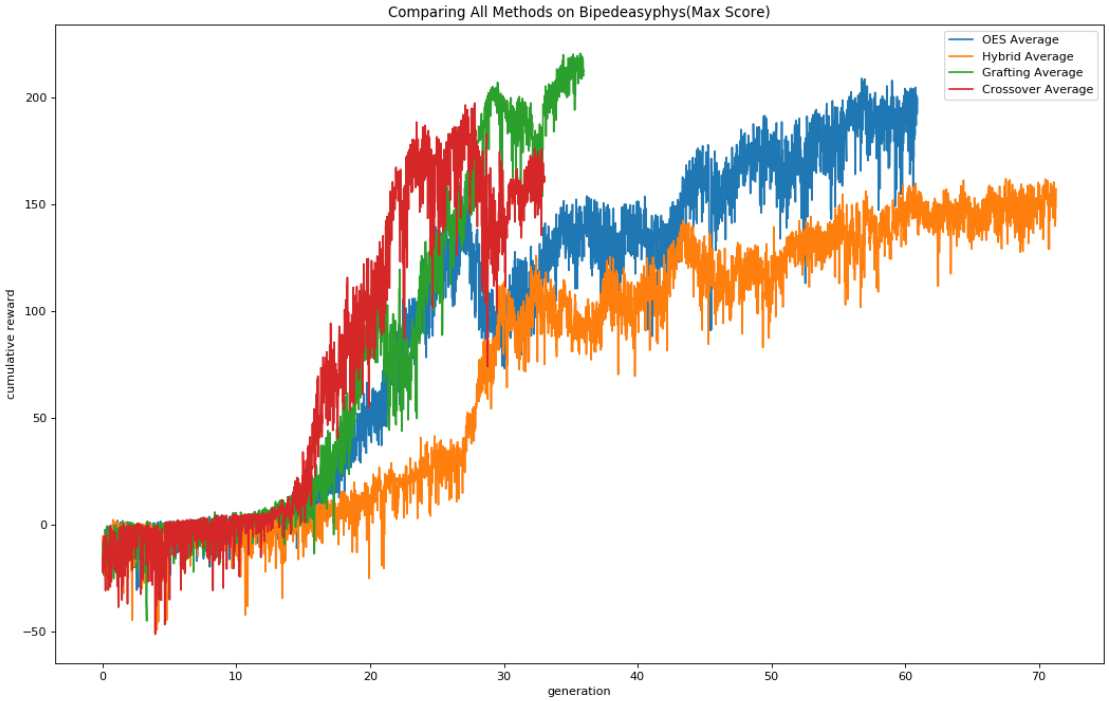
Following the results on the simpler problem, we began experimenting with the BipedEasyPhys experiments in order to see how the models perform when attempting co-design. This should be a much more difficult task to solve. To demonstrate this, we have first plotted the OpenAI model solving the BipedEasy task with the OpenAI model solving the BipedEasyPhys task.



We again begin by investigating if the Karl Sims models will be successful on the task.



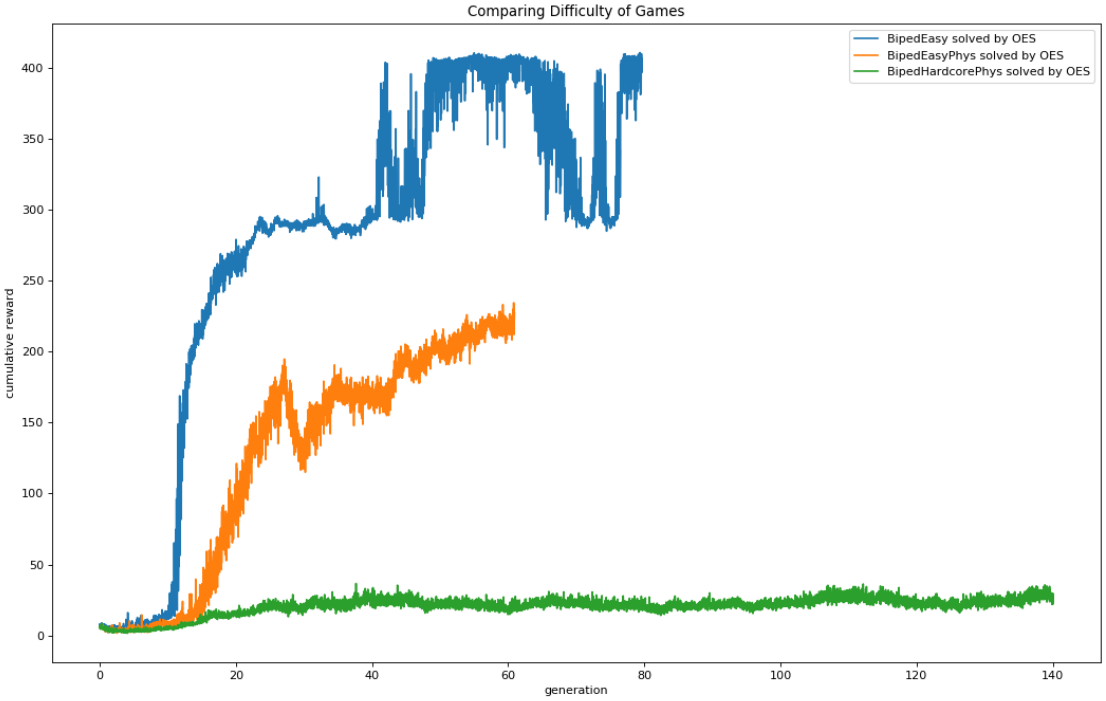
The models were successful on this task, however in this task the hybrid model did not perform better than the two stand alone methods. Possible reasons for this will be discussed in the following section.



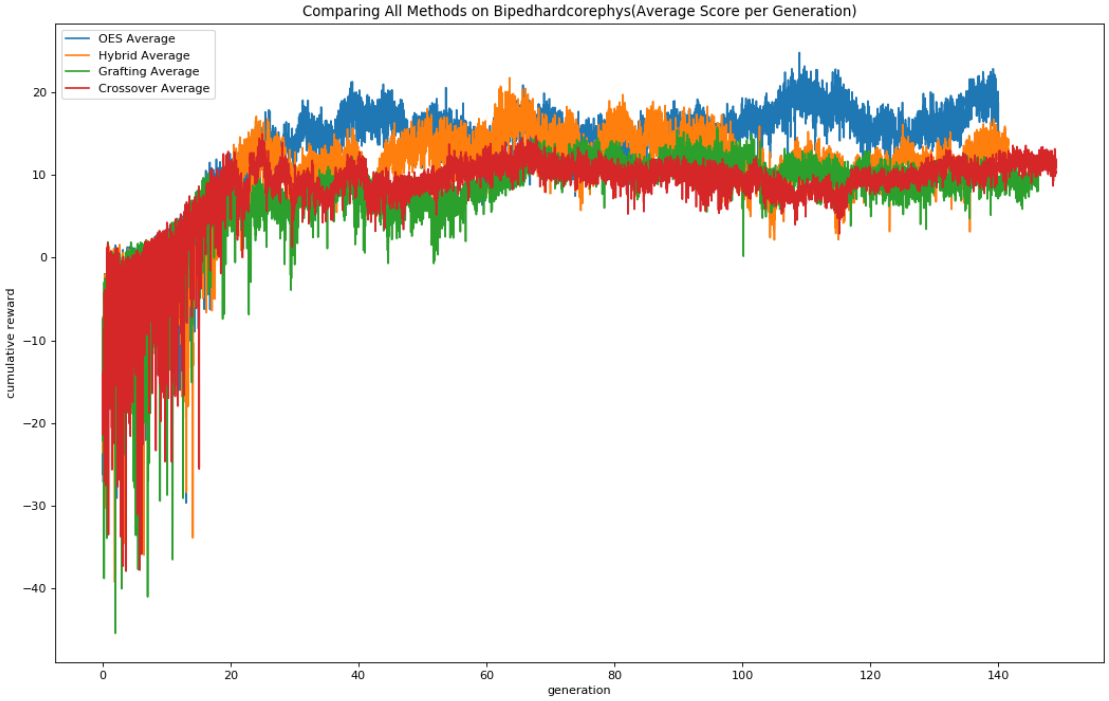
In this instance, again the Karl Sims models were able to out-perform the standard OpenAI models, with the exception of the Karl Sims Hybrid model.

4.3 BipedHardcorePhys

We then attempted to solve the BipedHardcorePhys task with our models. As seen in the following graph, the OES model was unsuccessful in making meaningful progress even after running for twice as long as it had run on the other tasks.



The description for the BipedalWalker tasks mentions how the bipedeasy task may take a day or two to complete, but the bipedhardcore may take weeks to solve. Due to the ESTool architecture not being designed for training models for this prolonged a period of time (we found it highly prone to crashes after 12-16 hours), and that adding the physical design components makes the task much more complicated, we did not have high expectations. Regardless, we ran all models for approximately a day on the problem.



As anticipated, the models were unable to achieve high scores on this task. While in this run it does seem the OpenAI standard model was able to out-perform the Karl Sims models, the scores achieved are so low that this may not be enough evidence to decide this. Further discussion on this will be included in the following section

Chapter 5

Discussion

5.1 Effects of increasing complexity

Temporarily not taking into account the effects of the BipedHardcorePhys task, it appears as if the more complex a task becomes, the more benefit the OES model receives from having the addition of the Karl Sims methods.

On the BipedEasy task the standard OES model was able to be trained more quickly than the Karl Sims models, but on the BipedEasyPhys task the Karl Sims models performed better. This indicates that when the complexity was increased due to the added burden of having to optimize the physical morphology of the robot, the addition of the Karl Sim's methods are more beneficial for OES.

This could be explained by the discussion in Karl Sims' paper regarding the trade-off between control and complexity[3]. When working on an optimization problem such as this there is a spectrum how constrained the system is, in which if the user gives up some amounts of control of the design the system become more capable of "automating the creation of complexity". In the simpler Bipedeasy task, the model is not required to generate as much complexity as other tasks, and therefore does not receive much benefit in being able to generate more complexity. However, when co-design is introduced in the Bipedeasyphys task and a substantial increase in the necessary complexity occurs, then the Karl Sims models are able to benefit from their loss of control.

While this increased performance did not manifest in the BipedHardcorePhys experiments, this may be explained by other factors, which will be expanded on later in this report.

5.2 Hybrid Model Discussion

In the initial easier task, the hybrid model found more success than the models which just used the cross-over or grafting mutations. However when the complexity was increased in the Bipedeasyphys this was no longer the case.

This is a somewhat mysterious behavior. As far as the code is concerned, the hybrid model functions by taking the breeding results which would occur for each stand-alone model, but only taking the first half generated by each.

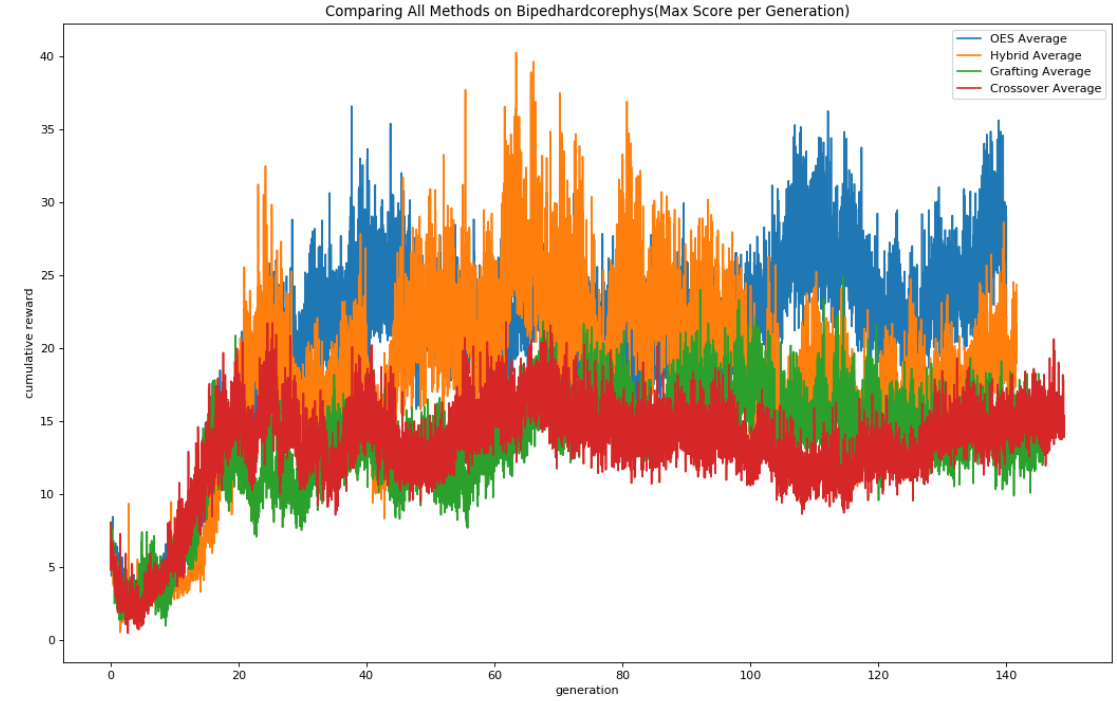
Initial investigations into this began by debugging the code, but no obvious bugs were discovered. This could have been expected however, as based on the results the hybrid model was actually successful in solving the task, just not as as successful as the other models.

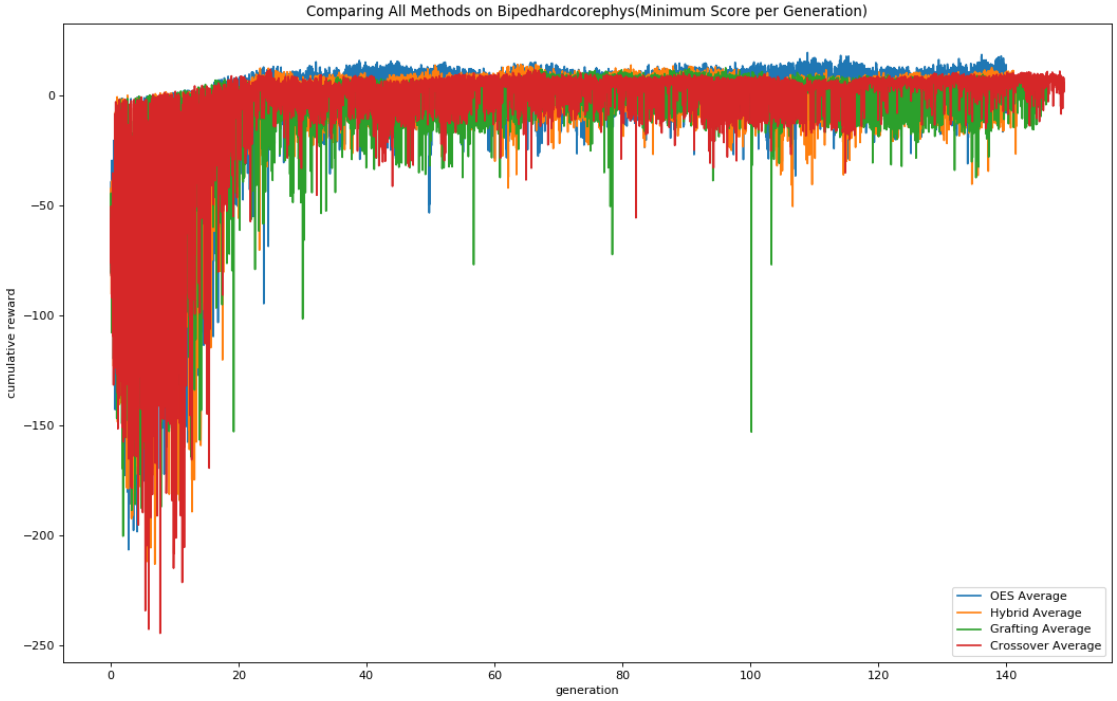
A potential explanation could be found in the same explanation as to why the other two Karl Sims models performed better than the OES model. Perhaps in some way, using two cross-breeding methods instead of one somehow reduces the variety created, so as the tasks become more complicated the model's performance falls off. This wouldn't be the expected case as one would assume that using 2 methods instead of 1 would produce MORE variety not LESS, but more investigation on a wider variety of tasks would be needed to know for sure.

5.3 BipedHardCorePhys Discussion

As seen in the figures, none of our models were able to build a successful solution to the BipedHardCorePhys task. We believe this largely to be due to the computing power we had access to for these experiments.

We do not believe that the model's had successful enough results on this task for it to be relevant to discussion on the benefits of the Karl Sims methods. To support this, we can investigate the results of the model, but instead plotting the maximum and minimum scores per each generation.





Based on these graphs, the results seem much like random noise due to the models

struggling to get a foot-hold on the problem. Therefore, further experimentation on this task with improved hardware would be necessary for relevant results.

5.4 Comparison to Reinforcement Learning

In OpenAI's work with OES, they found that their methods are competitive with reinforcement learning techniques, and are able to scale well with additional computational power[2]. These comparisons can again be seen from the figure in section 1.3.

Therefore further improvements on OES must be at least competitive with reinforcement learning, and if not, superior.

Even in the former case, having genetic algorithms which are competitive with reinforcement learning techniques for more complicated tasks is significant. As mentioned in the section Genetic Algorithm Advantages for Co-Design, there are a large number of benefits to genetic algorithms in comparison to reinforcement learning. To re-summarize the points from section 2.2:

Genetic Algorithm Benefits for Non-Supervised Learning

- No Back Propagation
- Parallel Training
- Higher Robustness
- Better suited for long episode times

Chapter 6

Conclusions

Robotic co-design is a difficult task for any model to train on successfully. While reinforcement learning methods are widely used in this space, genetic algorithms are a favourable alternative. OpenAI's OES is a genetic algorithm based model which is competitive with reinforcement learning techniques, but can struggle as tasks become more complicated. This is potentially due to OES's mutations not producing enough variety.

Karl Sims developed cross-breeding mutations which produce a significant amount of variety. Therefore, our research was in improving OES by utilizing these aforementioned Karl Sims mutations.

After our experimentation on the Bipedal Walker tasks, we have found that as tasks become more complicated, OES gains benefit from the addition of Karl Sims mutations.

This is notable, as this shows promise in application to the highly complicated task of co-design for robotic design automation. This would be a tremendous alternative to machine learning based methods, which genetic algorithms have several significant advantages over. These advantages include no back propagation, parallel training capabilities, increased robustness, and better performance in tasks with long episode times.

With the promising results of our OES and Karl Sims cross-bred models on these tasks, it would be worth investigating into how this merged model performs on other non-supervised learning tasks.

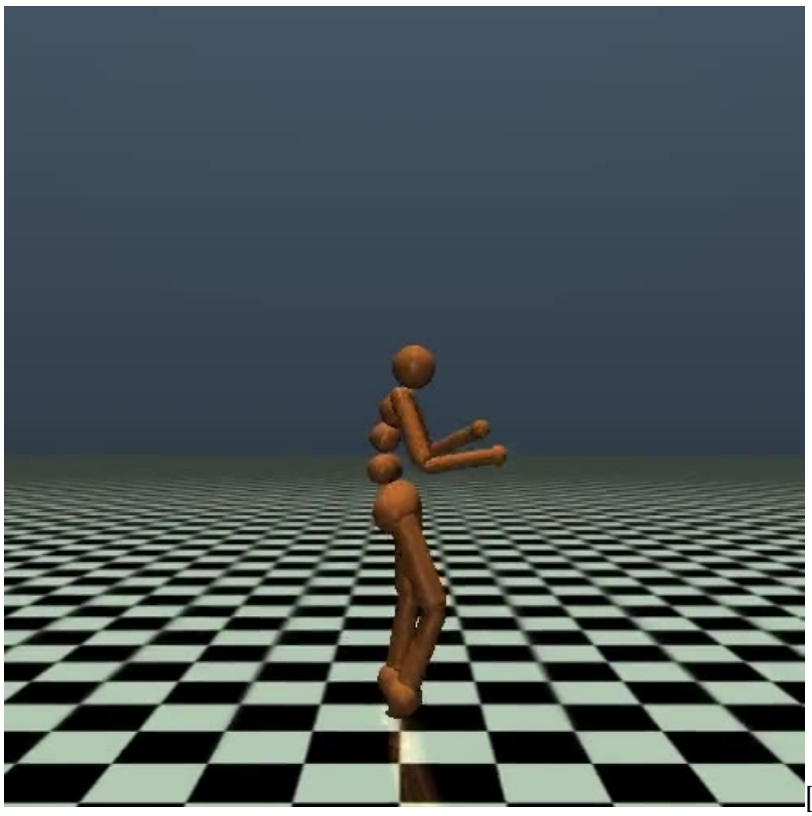
Chapter 7

Future Work

7.1 Experimentation on Increasingly Complex Tasks in Co-Design

While it appears from our experiments that OES benefits from Karl Sims methods as tasks become more complex, more investigation would be needed to know for sure. We attempted to generate more data on this by running experiments on the BipedHardcorePhys task, but with the computing power we had available this was not an option.

Therefore, it would be of interest to run these experiments again on hardware capable of solving these tasks, and seeing how the learning rates of the various models would perform. Furthermore, investigation into 3D models would be relevant. MujoCo is a 3D physics engine which, much like Box2D, has a variety of tasks created for OpenAI's gym[20].

 [4]

Amongst their tasks is another Bipedal walker, but in 3D. This biped has many more physical characteristics than it's Box2D counterpart. Modifying this task (in the same way we modified the Box2D version) such that the model has to optimize the physical characteristics as well as the motor controllers (therefore engaging in co-design) would be an excellent task to further test the modified OES models.

7.2 Investigation Into Tasks Outside of Robotics

Beyond Co-Design, genetic algorithms have a number of potential applications. For the most part, any task which can be accomplished via reinforcement learning techniques could be attempted by genetic algorithms. As noted by OpenAI in their work, all of the advantages which we have listed in section 2.3 of this paper do not just apply to Co-Design, but to any unsupervised learning problem[2].

To this point, an investigation into the performance of OES and the Karl Sims modified OES models into a broad spectrum of reinforcement learning applications would be an interesting space of research. As an example of this, there are projects attempting to use reinforcement learning to optimize trading algorithms for various assets, such as FOREX[21] or Crypto[22]. As discussed in 2.2, genetic algorithms are more robust, and better suited for long episode times. These two factors may enable genetic algorithms to out-perform reinforcement learning techniques in complex economic tasks. However, as of the writing of this work, there is very limited research in this space.

Bibliography

[1] R. Desai, B. Li, Y. Yuan, and S. Coros, "Interactive co-design of form and function for legged robots using the adjoint method," *CoRR*, vol. abs/1801.00385, 2018.

[2] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," 2017.

[3] K. Sims, "Evolving virtual creatures," in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, (New York, NY, USA), p. 15–22, Association for Computing Machinery, 1994.

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[5] Reportlinker, Nov 2020.

[6] J. Hiller and H. Lipson, "Automatic design and manufacture of soft robots," *IEEE Transactions on Robotics - TRob*, vol. 28, pp. 457–466, 04 2012.

[7] A. Hassan and M. Abomoharam, "Modeling and design optimization of a robot gripper mechanism," *Robotics and Computer-Integrated Manufacturing*, vol. 46, pp. 94–103, 2017.

[8] L. Carlone and C. Pinciroli, "Robot co-design: Beyond the monotone case," *CoRR*, vol. abs/1902.05880, 2019.

[9] A. Censi, "A mathematical theory of co-design," 12 2015.

[10] A. Sapietová, M. Sága, I. Kuric, and Václav, "Application of optimization algorithms for robot systems designing," *International Journal of Advanced Robotic Systems*, vol. 15, no. 1, p. 172988141775415, 2018.

[11] H. A. Pierson and M. S. Gashler, "Deep learning in robotics: a review of recent research," *Advanced Robotics*, vol. 31, no. 16, p. 821–835, 2017.

[12] A. M. TURING, "I.—COMPUTING MACHINERY AND INTELLIGENCE," *Mind*, vol. LIX, pp. 433–460, 10 1950.

[13] N. A. Barricelli, "Numerical testing of evolution theories," *Acta Biotheoretica*, vol. 16, pp. 69–98, Mar 1962.

[14] Mike, "What is a genetic algorithm?," Jun 2019.

[15] M. Schoenauer, "Evolutionary computation," 12 1998.

[16] R. Al-falluji and A. Al-Zoghabi, "Genetic algorithms genetic algorithms agenda," 04 2019.

[17] J. Pollack, H. Lipson, P. Funes, S. Ficici, and G. Hornby, "Coevolutionary robotics," pp. 208–216, 02 1999.

[18] E. Catto.

[19] D. Ha, "Evolving stable strategies," *blog.otoro.net*, 2017.

[20] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control.," in *IROS*, pp. 5026–5033, IEEE, 2012.

[21] J. Carapuço, R. Neves, and N. Horta, "Reinforcement learning applied to forex trading," *Applied Soft Computing*, vol. 73, 09 2018.

[22] D. Barra, R. Feltrin, H. Almeida, and S. Marin, "Reinforcement learning applied to a cryptocurrency portfolio in a complexity environment," *Revista Economia Ensaios*, vol. 36, p. 28, 01 2021.