

https://dspace.org/

Augmentative topology agents for open-ended learning

Nasir, Muhammad Umair

2024
https://hdl.handle.net/10539/37468

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

Augmentative Topology Agents For Open-ended Learning

Muhammad Umair Nasir

Supervisor(s):
Prof. Christopher W. Cleghorn
Dr. Steven D. James



A research report proposal submitted in partial fulfillment of the requirements for the degree of Master of Science in the field of Artificial Intelligence

in the

School of Computer Science and Applied Mathematics
University of the Witwatersrand, Johannesburg

10 June 2023

Declaration

I, Muhammad Umair Nasir, declare that this proposal is my own, unaided work. It is being submitted for the degree of Master of Science in the field of Artificial Intelligence at the University of the Witwatersrand, Johannesburg. It has not been submitted for any degree or examination at any other university.



Muhammad Umair Nasir

10 June 2023

Abstract

We tackle the problem of open-ended learning by improving a method that simultaneously evolves agents and increasingly challenging environments. Unlike previous open-ended approaches that optimize agents using a fixed neural network topology, we hypothesize that the open-endedness and generalization can be improved by allowing agents' controllers to become more complex as they encounter more difficult environments. Our method, Augmentative Topology EPOET (ATEP), extends the Enhanced Paired Open-ended Trailblazer (EPOET) algorithm by allowing agents to evolve their own neural network structures over time, adding complexity and capacity as necessary. Empirical results demonstrate that ATEP results in open-ended and general agents capable of solving more environments than a fixed-topology baseline. We also investigate mechanisms for transferring agents between environments and find that a species-based approach further improves the open-endedness and generalization of agents.

Acknowledgements

First and foremost I would like to acknowledge the prompt and tireless help of my supervisor, Prof. Christopher Cleghorn, who has guided me, in a way that was very easy for me to grasp, and have calmly listened to many of my impractical ideas to produce a very practical one. Every single meeting was a progress towards a more focused and cutting-edge idea. These few lines do not comprehend my acknowledgment for Prof. Cleghorn as he does not only guide me with respect to thesis but also keeps the discussion in a way which would lead to advices about future career directions.

Secondly I would like to acknowledge the efforts of my co-supervisor, Dr. Steve James. In a very short time of interaction, he has proved to have helped a lot by navigating me towards the right direction with respect to this thesis, and research mentality as well. I am looking forward to having a great experience from both of the supervisors this year.

I would like to extend my special thanks to my parents and my wife, who have provided me with unconditional support during the ups and downs from the start of the degree. Lastly, I will extend my acknowledgements to my son, He always comes by me when I am working to cheer me up and to discuss ideas as well.

Contents

Declaration i

Abstract ii

Acknowledgements iii

List of Figures vi

List of Tables ix

1 Introduction 1
- 1.1 Introduction to Open-ended Learning . . . 1
- 1.2 Problem Statement . . . 3
- 1.3 Research Questions, Aims, and Objectives . . . 4
  - Research Questions . . . 4
  - Research Aims . . . 4
  - Research objectives . . . 4
- 1.4 Overview . . . 5

2 Background 6
- 2.1 Machine Learning . . . 6
- 2.2 Neural Networks . . . 7
- 2.3 Reinforcement Learning . . . 9
- 2.4 Genetic Algorithms . . . 10
- 2.5 Neuroevolution . . . 12
- 2.6 Neuroevolution of Augmenting Topologies . . . 14
- 2.7 Open-ended Learning . . . 18
- 2.8 POET and Enhanced POET . . . 19
- 2.9 Related Work . . . 24

3 Research Methodology **27**
3.1 Open-endedly evolving the topology of agents 27
3.1.1 Augmentative Topology Enhanced POET 27
Fitness-Based Transfer 29
Species-Based Transfer 30
3.1.2 Experimental setup 31
Environments 31
Agent 32
EPOET Framework 33
ATEP Variants and Baselines 33
Hyperparameter Selection 34

4 Results and Discussion **35**
4.1 Open-endedness 36
4.2 Nodes complexity exploration 37
4.3 Generalization Ability 42

5 Conclusion And Future Directions **45**

Bibliography **47**

A Hyperparameter settings **58**

List of Figures

2.1 This diagram illustrates a simple NN, also called a perceptron. Yellow nodes represent inputs that are multiplied by the weights of the connection it corresponds to. These are then summed together and passed to an activation function f which produces the output. 8
2.2 This diagram illustrates a fully connected Deep NN with two hidden layers. 8
2.3 A general Reinforcement Learning algorithm flow. An agent takes an action, for which it gets a reward and a different environment state. It repeats till the episode is over. 11
2.4 Different types of operators that can be used in a GA [47] 13
2.5 Representation of Genotype and Phenotype of NEAT individual [102]. Node genes display all nodes in the network. Connection genes store all information about the connection, such as in and out nodes, the weight of the connection, if it is enabled or disabled, and the innovation number. 15
2.6 The diagram illustrates structural mutation with the addition of connection and nodes [102]. Connection genes represent the mutations. The genotype, which is above the phenotype (or structure of the network), starts with an innovation number. An innovation number is a historical marker that is used to identify the ancestors of the node. The genotype then contains a connection between nodes. In *add connection* mutation, we add a connection which then gets an incrementally new innovation number. For *add nodes* mutation, we first disable the connection $3- > 5$ then add a new node 6, which will add two new genes 8 and 9. 16

2.7 Figure displays crossover between two parent genomes and explains how similar, disjoint and excess genes work [102]. From similar genes, we will always select from the fitter parent. The disjoint and excess genes are always inherited as they are. Innovation number defines excess and disjoint genes. Disjoint means gene present within the range of maximum innovation number of the other parent and excess means out of the range of maximum innovation number present in the other parent genotype. 17
2.8 A view of the *2D Bipedal Walker Hardcore Environment* [115]. Possible obstacles are stumps, gaps, steps and roughness of the surface. 20
2.9 Figure illustrates the process to calculate PATA-EC score[114]. First, all active and archived agents are evaluated on the created environment *E*. The scores of the evaluations are clipped between a min-max range. The scores are ranked and then normalized. Thus, for one environment we get a vector of the length of the number of active and archived agents. 21
2.10 A CPPN generator (left) generating a level (right). [114] 22
2.11 Example environment produced by EPOET. [114] 23

3.1 A flowchart demonstrating the flow of the ATEP framework, with blocks in green being where ATEP differs from POET. For both EPOET and ATEP, each environment is associated with an agent, represented by an ES population for EPOET and a NEAT population for ATEP. The environment images used in the chart were created by ATEP. Refer to Appendix A for pseudocode describing the transfer mechanisms used in ATEP. 28
3.2 A two-legged agent with a hull from OpenAI Gym's 2D Bipedal Walker environments [8]. 32

4.1 Accumulated Number of Novel Environments Created and Solved (ANNECS). To save the compute, we stop the NT- and RT-ATEP experiments early, as it is clear that they perform poorly. The error bars are standard deviation, the line represents the mean and both are calculated over 5 runs. 35

4.2 Cumulative sum of the number of function evaluations, with the Y-axis, converted to a log-scale. While ATEP requires significantly more function evaluations, we find that its total wall-time is only 3 times more than EPOET, as the neural networks are generally smaller and each evaluation does not take as long. 37
4.3 Mapping of fitness to nodes. We plot values every 1000 iterations, starting at iteration number 150. Each dot represents a specific iteration, as well as the mean fitness overall environments and the mean number of nodes in the population. The distribution on top represents the distribution of nodes while the distribution on the left represents fitness distribution. 38
4.4 Analysis with respect to the number of nodes. Figure **(a)** shows FNR along iterations, **(b)** shows ANR along iterations, **(c)** shows the addition of nodes along iterations 39
4.5 From left to right: NNs at iterations 0 and 2000 of when it was created. 40
4.6 From left to right: NNs at iteration 4000 and 6000 of when it was created. 41
4.7 Figure **(A)** shows each algorithm being tested on the 20 latest environments created by all other algorithms, i.e., each algorithm is evaluated on 60 environments. The Y-axis shows the percentage of environments in each category. Each test is conducted for 30 runs and the mean scores are taken. Figure **(B)** shows the maximum score of the 30 runs. 42
4.8 Figures showing generalization capabilities. Figures **(a)**, **(b)** and **(c)** show agents of 80 solved environments being tested on all 80 environments, for EPOET40x40, FBT-ATEP, and SBT-ATEP respectively. Note that EPOET20x20 does not take part in this test as it failed to produce 80 environments in the run. 43
4.9 Action Distributions for (top row) SBT-ATEP, (middle row) FBT-ATEP, and (bottom row) EPOET40x40. Each column represents one specific dimension of the action array. 44

List of Tables

2.1 Environmental parameters for POET [115]. 19

A.1 EPOET general hyperparameter settings for ATEP 58
A.2 ES hyperparameter settings 58
A.3 NEAT hyperparameter settings 59
A.4 CPPN hyperparameter settings 60

Chapter 1

Introduction

1.1 Introduction to Open-ended Learning

Evolution has taken place for billions of years and is taking place as we speak. It has always been open-ended toward the goals it reached. Human evolution, from cavemen to trying to land on Mars, is open-ended as we did not think of landing on Mars when we were cavemen. We were not thinking of creating computers when we made vacuum tubes. There are many examples in human history alone, and not considering the evolution of the universe itself, which directs us to believe that we are goal-oriented beings but innovations happen open-endedly. We also witness nature's spectacular ability to invent through the diversity of every single living being. We see the diversity in each species separately, from plants to different animals to human beings as well, all of us have diversity within ourselves. This leads us to believe that diversity and innovation comes from an ongoing process that was not something in past but remains the same for 4 billion years, thus this ongoing creative process is called *open-ended evolution* [75].

The concept of open-ended evolution has been a part of the artificial life worlds (or ALife worlds) research field for approximately two decades. These worlds, like Tierra [84], Avida [71], Division Blocks [95], Polyworld [119] and Chromaria [94], have ALife creatures in them, and have goals like running around and being a predator or prey. Induction of the concept of open-ended evolution, in ALife, was necessary as this is how humans reached their peak of intelligence and nature, at the peak of diversity. ALife, creating worlds through open-ended evolution, has led researchers to think about how open-ended evolution or *open-endedness* can benefit the AI community. Stanley, Lehman, and Soros [101] have urged researchers to bring open-endedness in algorithms to achieve human-level intelligence by providing guidance on what must be the ingredients of these *open-ended algorithms* and

how should we expect them to behave. Stanley [99] points out the fact that open-endedness deserves a place side-by-side with Artificial Intelligence (AI) as one of the great computational challenges of the time. From AI's perspective, open-ended algorithms mean that the algorithms that are theoretically never-ending and the solution that needs to be solved complexify over time.

Open-ended evolution truly came into the field of AI when one of the *Topology and Weight Evolving Artificial Neural Networks* (TWEANNs) and *NeuroEvolution* (NE) methods [36, 39, 66, 116], named *NeuroEvolution of Augmenting Topologies* (NEAT), by Stanley and Miikkulainen [102] surfaced and gained popularity as it solved complex Reinforcement Learning (RL) [107] tasks very efficiently, through evolving topologies. This opened up ideas that we can indefinitely evolve Neural Networks (NNs). It was early in this research space and much was yet to be discovered and so Lehman, Stanley, et al. [56] introduced *Novelty Search* (NS) algorithm with the intent to drive the research community one step closer towards open-ended evolution. The intent behind the NS algorithm was that the fitness-based Evolutionary Algorithms (EAs) usually ignored paths that were promising but were not yielding good results at first. Novelty search rewarded agents for finding novel paths which were not explored before. This led to the emergence of new types of algorithms called *Quality Diversity* (QD) algorithms which include diversity with an objective sense of progress.

While QD has successfully been used in numerous domains, such as robotic locomotion [16, 68, 108], video game playing [26] and procedural content generation [50, 25, 5], it still is not completely open-ended. One reason for this is that the search space for phenotypical behaviour descriptors, i.e. defining a space of behaviours to search within, remains fixed [68]. This descriptor space is vast but limited, and at some point in time, the full spectrum of search space is sampled. This significantly depreciates the pressure towards searching more diverse solutions.

A way to circumvent the fixed behavioural descriptors is with a coevolutionary [76] approach which is the focus of our research. The coevolution brings the properties of multi-agent interactions as two or more populations to evolve and interact with each other in a cooperative [117] or competitive [31] dynamics. Still, the co-evolution will not bring open-endedness if the goals remain static. Thus Minimal Criterion Coevolution (MCC) [7] opened up a new path by coevolving problems and solutions. Here, MCC evolved mazes as problems and agents as solutions. This enabled new environments to emerge from previous ones and new agents

that would solve them. This ongoing coupled interaction gave the possibility of open-endedness. MCC introduced a new kind of coevolution where the populations can evolve only when meeting a minimum criterion with respect to the other population. The maze solver (a neural network) has the minimum criteria to solve the maze and a maze has the minimum criteria to be at least solved by one maze solver. This leads to the continuous evolution of mazes and maze solvers without behaviour characterization, which is a need for QD algorithms.

However, MCC had some limits; for instance, it only allows new problems if they are solvable by individuals in the current population. This leads to only slight increases in difficulty, and complexity which only arise randomly. Taking this into account, *Paired Open-ended Trailblazer* (POET) [115] builds upon MCC, but instead allows the existence of unsolvable environments, if it was likely that some individuals could quickly learn to solve these environments. POET further innovates by transferring agents between different environments, to increase the likelihood of solving hard problems. While POET obtained state-of-the-art results, its diversity slows down as it evolves for longer. Enhanced POET [114] adds improved algorithmic components to the base POET method, resulting in superior performance and less stagnation. Enhanced POET, however, uses agents with fixed topology neural network controllers. While this approach works well for simple environments, it has an eventual limit on the complexity of tasks it can solve: at some point of complexity, the fixed topology agents may not have sufficient capacity to solve the environments.

To address this issue, we propose *Augmentative Topology Enhanced POET* (ATEP), which uses NEAT to evolve agents with variable, and potentially unbounded, network topologies. We argue that fixed-topology agents will cease to solve environments after a certain level of complexity and empirically show that ATEP outperforms Enhanced POET (EPOET) in a standard benchmark domain. Finally, we find that using NEAT results in improved exploration and better generalization compared to Enhanced POET.

1.2 Problem Statement

Existing methods [44, 110, 114] have been implemented on a fixed topology with optimizers like Evolutionary Strategies (ES) [89], Maximum a Posteriori Optimization (MPO) [1], Proximal Policy Optimization (PPO) [91] etc. We see a problem here

that eventually, at some point of complexity, these fixed topology agents' will create delays to solve the environment. For a true open-ended, we need to have agents that continuously augment their topologies, which will complexify as the environments complexify. Agents with fixed topology tend to stagnate as the environment complexifies, which is observed in EPOET.

1.3 Research Questions, Aims, and Objectives

Research Questions

By extending our problem statement defined in section 1.2, we ask the question: can agents that augment their topologies learn endlessly through the framework of EPOET? Learning endlessly is the essence of Open-ended Learning algorithms, which implies that it should not stop learning when environments are highly complex. EPOET has introduced a measure to quantify open-endedness which we will discuss in later sections.

The second question that we ask is can we use a different transfer mechanism that aids more open-endedness and produces better-generalized agents? By generalization, we mean agents that can solve some environments with high rewards while they can solve some environments with achieving minimum reward criteria to solve environments and are not catastrophically failing in any environment.

Research Aims

We aim to investigate how augmentative topology neural networks improve agents' behaviour toward increasingly complex environments. This research also aims to provide better generalization capabilities through augmentative topology agents.

Research objectives

The objectives of the research are to improve EPOET through an algorithm that augments the topology of the agent's neural network, such as Neuroevolution of Augmenting Topologies (NEAT). Another objective is to improve open-endedness and generalization capabilities through transfers that are specifically derived through the qualities of an algorithm that augments the topologies of agents.

1.4 Overview

In chapter 2, we will discuss all underlying algorithms and concepts that are needed to understand our research in depth. Our research methodology will be discussed in chapter 3, where we will discuss all the components of our proposed algorithm and the experimental setup for it. It will be followed by chapter 4 which will be about results and discussion on results. In the last chapter, chapter 5, we will end our research by concluding our work and by giving some possible directions for future works.

Chapter 2

Background

In this chapter, we will discuss the underlying algorithms that are necessary for our research. We will go into the depth of the original variant of the algorithm but we will leave the other variants for the readers to explore. We will briefly look into the concept of Machine Learning, Neural Networks, and Reinforcement Learning. We will then explain the Genetic Algorithms that make the basis of NEAT. We will then move forward to NEAT as we use it as the optimizing algorithm. Before looking into POET and Enhanced POET (EPOET), we will briefly go through the concepts of Open-ended Learning.

2.1 Machine Learning

Machine Learning (ML) [64] is a field of Artificial Intelligence (AI) [118] that uses data and experience to learn a solution to a problem. ML is not programmed to find an explicit solution but to find a solution based on trial and error. This results in diverse solutions from one ML algorithm. ML has been applied to various domains, such as predicting stocks [79], classifying images [58], translating from one language to another [96], learning how to play games [40], folding proteins [46], and many more.

All application domains may be vastly diverse, but the mechanism of learning for an ML model remains exceedingly similar. For the ML algorithm, one iteration will mean one look at the data available. This available data is called training data. Over the course of iterations, the algorithm improves its performance by tweaking the proposed solution to build a model that reaches the desired goal. The tweaking is done through a loss function that calculates how far the proposed solution is from the desired solution. Traditionally, we separate testing data that the model has not seen and examine how well the trained model performs on it.

ML approaches are divided into three main categories: Supervised learning, unsupervised learning, and reinforcement learning. The difference depends on how the model is being trained. **Supervised Learning** is a technique where the training data available has desired outputs as well [10]. These outputs are known as labels. The model trains to predict the label, thus improving upon it. **Unsupervised Learning** is a technique where we have data but no output labels [11]. The model has to learn patterns in the data to separate groups of data that should belong under a single label. **Reinforcement Learning** is a technique where the algorithm learns through interaction with the environment [106]. We will look into it in more detail in an upcoming section as it is one of the basic concepts needed for our research.

2.2 Neural Networks

Neural Networks (NN), also known as Artificial Neural Networks, are computing systems that are motivated by the human brain [43]. Like the human brain, NN has neurons that are called nodes in computational sciences. These nodes are connected through weighted connections, that resemble synapses of the human brain. In the human brain, an electric signal passes from one neuron to another to process a function. To replicate this process, NNs pass numerical values through connections.

Figure 2.1 shows a simple perceptron. Perceptrons are the building blocks of NNs. The output value of a perceptron is calculated by adding the weighted input vector and passing it to an activation function. This simple perceptron can act as a linear classifier where it can classify if this vector belongs to one category or another. A collection of these perceptrons are called Multi-layer Perceptrons (MLP) or NNs. Figure 2.2 shows a fully connected NN with two hidden layers. These hidden layers can be scaled to any number.

Similar to perceptron, each node will receive an input vector, which will be summed and passed to an activation function. This numerical value will now be inserted into another vector which will then do the same process as described until the last node. This output node will produce an output value for us. These NNs are sometimes called Deep Neural Networks as they can have multiple hidden nodes.

Deep Neural Networks are used in a sub-field of Machine Learning called *Deep Learning* (DL). DL is used for all kinds of ML problems, such as supervised learning, unsupervised learning, and reinforcement learning. There are different types of architectures, such as convolutional neural networks [73] and transformers [112], that

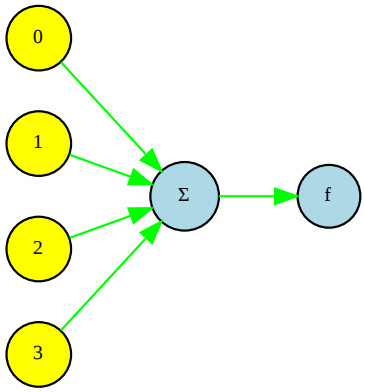


FIGURE 2.1: This diagram illustrates a simple NN, also called a perceptron. Yellow nodes represent inputs that are multiplied by the weights of the connection it corresponds to. These are then summed together and passed to an activation function f which produces the output.

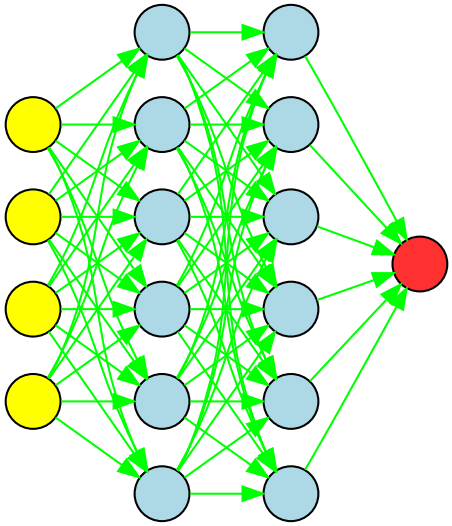


FIGURE 2.2: This diagram illustrates a fully connected Deep NN with two hidden layers.

are successfully used for tasks like object detection in an image or video [45], translation from one language to another [96], protein folding [46] and beating world champions in complex board games like Go [37].

Conventionally, DL uses gradient descent [86] and backpropagation [85] to improve the prediction of the NN. Gradient descent optimizes the weights of an NN by incrementally finding the right weight combination of the NN, while backpropagation is used to find the gradient of the loss function. The loss function, as discussed in the Machine Learning section, is used to determine how incorrect the model is. Incrementally, the loss function is moved towards the least incorrect model. Loss functions are set to be differentiable to make backpropagation work.

NNs uses activation functions which keep the values in a desired range to get a desired output. One such activation function is the hyperbolic tangent function (tanh). The tanh function takes a real-valued number and squashes it to the range between -1 and 1. In other words, it maps the input to a value between -1 and 1. It is an S-shaped curve but is symmetric around the origin (i.e., (0,0)). We explain tanh as we use it in this research. Mathematically, the function is defined as:

$$tanh(x) = (e^{x} - e^{-x}) / (e^{x} + e^{-x}) \tag{2.1}$$

Similar to other activation functions, tanh is non-linear. This property allows neural networks to learn and represent complex patterns. the tanh function is zero-centred, making it easier for models to learn in some cases, as it can output both positive and negative values, which can be more informative for the subsequent layers. The derivative of tanh (used during backpropagation) is easier to compute than many other functions. The derivative is $1 - tanh^{2}(x)$, which is always positive and varies from 0 (for input values far from 0 where the function saturates at -1 or 1) to 1 (at x=0). Thus, with the above-mentioned properties tanh becomes a powerful activation function to use in our research.

2.3 Reinforcement Learning

Reinforcement Learning (RL) is an ML technique for an algorithm to learn through experience in an environment and interaction with it [106]. In RL, an ML model is known as an agent, that interacts with the environment in which it is present, with

the goal to maximize total rewards. Rewards are calculated through a reward function that decides which action will produce a positive reward or a negative reward which is known as regret. An agent interacting with an environment is generally modelled as a Markov Decision Process (MDP) [80]. Mathematically, an MDP is a control process with discrete-time steps. An MDP consists of the following:

- A set of actions, A, taken by the agent.
- A set of environment and agent states, S.
- $P_a(s, s^{'}) = P(s_{t+1} = s^{'} | s_t = s, a_t = a))$, where P_a is the probability of transition at time t from stat s to $s^{'}$, with the action a taken at time t
- A reward $R_a(s, s^{'})$ after an action a taken to transit from s to $s^{'}$.

Essentially, the above steps can be summarized such that at each time step t, an agent is on a current stat s_t with a reward r_t. The agent then chooses the next action a_t from the possible actions it can take, which is then sent to the environment. The environment progresses to a new state s_{t+1} with the reward r_{t+1}. Figure 2.3 shows a general RL algorithm flow. RL algorithms that use Neural Networks, as optimizers, are called Deep RL [32]. Deep RL has been used for many complex applications, such as autonomous driving [52], beating world champions in the game of Go [37], creating video game bots that can generally play games [40], and generating algorithms for faster matrix multiplications [30].

2.4 Genetic Algorithms

In recent years, metaheuristic algorithms [20] have been used to solve real-life complex problems in various domains of engineering [53]. A metaheuristic algorithm is an algorithm that attempts to find the most feasible solution out of all possible solutions to a possible optimization problem [20]. Most metaheuristic algorithms are inspired by real-life biological evolutionary processes and swarm behaviors of many different species, usually of birds, flies, and insects [6]. These metaheuristic algorithms are categorized into two main categories: Single-solution-based and Population-based. Some well-known single-solution-based metaheuristic algorithms are Tabu Search [35] and guided local search [113]. However, single-solution-based algorithms are prone to get stuck in local optima. Population-based

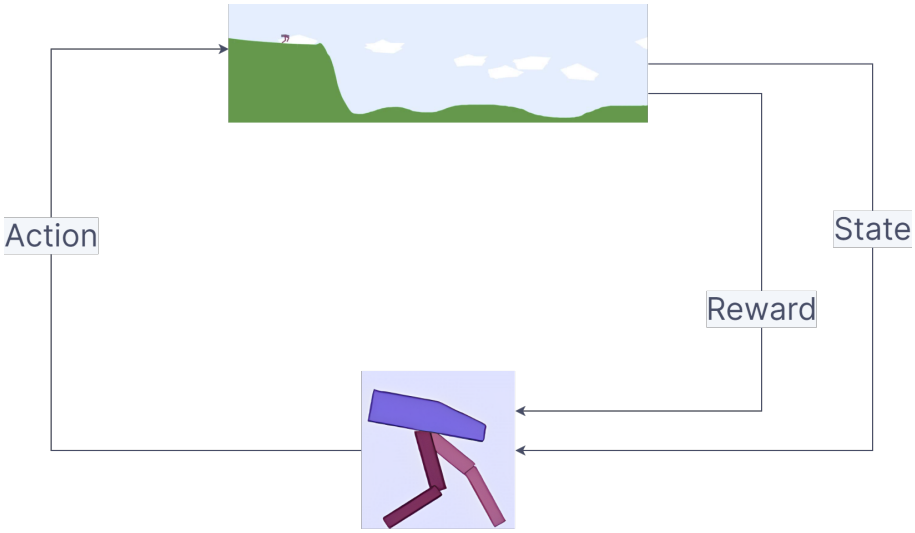


FIGURE 2.3: A general Reinforcement Learning algorithm flow. An agent takes an action, for which it gets a reward and a different environment state. It repeats till the episode is over.

metaheuristic algorithms utilize multiple candidate solutions to search for global optima. Well-know population-based algorithms are Genetic Algorithms (GAs) [42], Particle Swarm Optimization [48] and Ant Colony Optimization [21]. We will go into the depth of GAs as it is important for our research.

GAs are inspired by natural selection [27] and emulate evolution and thus are among a field of study, called **Evolutionary Algorithms** or **Evolutionary Computation** [3]. Each chromosome (individual) passes through evolutionary operators, such as selection, crossover, and mutation, to improve a fitness function [63], where the strongest will survive and the weakest will be eliminated. A fitness score derived from a fitness function will decide how good an individual chromosome is. In selection, the chromosomes are selected on the basis of their fitness score for further processing. In the crossover operator, a random set of chromosomes are chosen and it changes the subsequences between chromosomes to create offsprings. In mutation, some bits of the chromosomes will be randomly flipped, which could be other techniques such as bits being displaced, on the basis of probability [3, 41, 120]. Another important factor to set up a GA is the representation of the chromosome, which is called chromosome encoding.

Algorithm 1: Genetic Algorithm

Input : Population size N, the maximum number of iterations MaxIter and fitness function f(.)

Generate random initial population P_k (k=1,2,3,...,N);
Compute fitness $f(P_k)$
foreach $i < maxIter$ **do**
  foreach n *in* N **do**
    C_1,C_2 = Selection();
    O_1 = Crossover(P_1, P_2, CrossoverProbability);
    O_1 = Crossover(O_1, MutationProbability);
    NewPopulation = Add(O_1);
  end
  NewPopulation = ReplaceOldPopulation();
end

An example of the procedure of GAs is as follows: A population (N) of n chromosomes is initialized randomly. The fitness of each chromosome in X is computed. Then two chromosomes from the population X are selected, let's say C_1 and C_2 according to their fitness value. A crossover operator, such as single-point crossover (or any other variant) is applied on both of the selected chromosomes with a crossover probability, to produce a new chromosome, called offspring O_1. Then we apply a mutation operator, such as uniform mutation operator [83], on O_1 with some mutation probability. The mutated offspring is placed in the new population. This selection, crossover, and mutation operators will be repeated on the current population until the new population is complete, which will then be replaced by the old population. Algorithm 1 explains the flow of a GA. There are many types of chromosome encoding, selection, mutation, and crossover operators. Figure 2.4 shows different types of operators a GA can use, this is not an exhaustive list.

2.5 Neuroevolution

NeuroEvolution (NE) is a field in Evolutionary Algorithms (EAs) that applies EAs to Neural Networks (NNs) [33]. The goal of NE is to optimize the connection

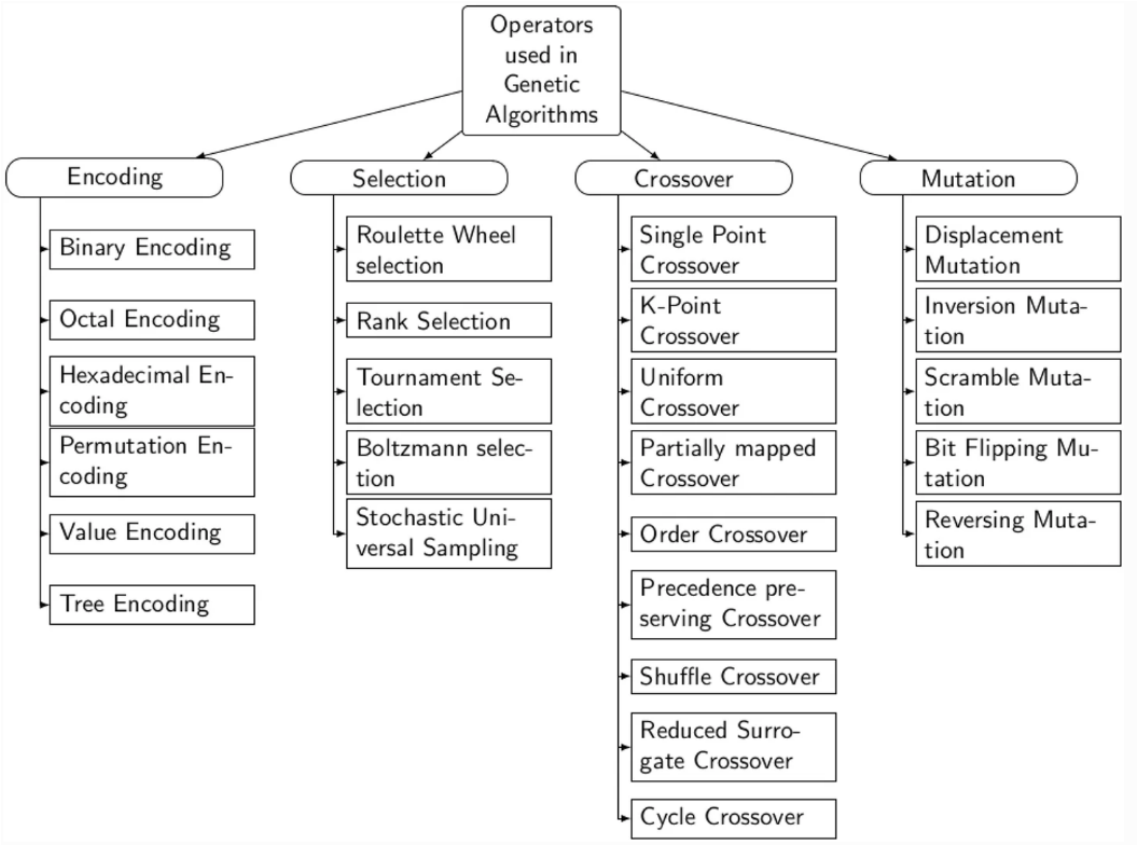


FIGURE 2.4: Different types of operators that can be used in a GA [47]

weights and topology of Neural Networks (NN). Similar to EAs, NE uses evolutionary operators like selection, mutation and crossover, to produce better individuals in the population. Similarly, the performance of these individuals are decided through a fitness score that is calculated through a fitness function. For example, if we use Genetic Algorithm to optimize the weights of an NN, the flow of the algorithm will remain the same as Algorithm 1, just the encoding will be different.

Genetic algorithms [105], Evolutionary Strategies [89], Genetic Programming [111], and Differential Evolution [103] are some of many EAs that have been used for optimizing weights for NE, where topologies remain fixed. Topology and Weight Evolving Artificial Neural Networks (TWEANNs) [67, 93, 98] use these EAs with a different genetic encoding. GAs has been used with a bit string representation, has been used with a graphical encoding structure, some have used only mutation operators and argued that the crossover of topologies only has a negative impact and leads to a loss of functionality, which we will discuss in the upcoming section.

Some have used indirect encoding [24], which means the genomes are represented by another program so that we can have more expressive encoding.

NE has been successfully used any many fields such as reinforcement learning [89], evolutionary robotics [15], procedural content generation for video games [60], etc.

2.6 Neuroevolution of Augmenting Topologies

We will now explain the NeuroEvolution of Augmenting Topologies (NEAT) as it is the underlying optimizing algorithm for our research. NEAT takes inspiration from GA's ability to use evolutionary operators and applied them to neural networks. NEAT revolutionized the field of NeuroEvolution (NE) by introducing a few distinctive features. NEAT starts off with a population of basic NN architectures, i.e, input neurons connected with output neurons without any hidden layers. The individual (chromosome) and also referred to as genome (or genotype) will consist of all information regarding nodes and connections. Figure 2.7 illustrates what a genome looks like and how the neural network (or the phenotype) will eventually be placed.

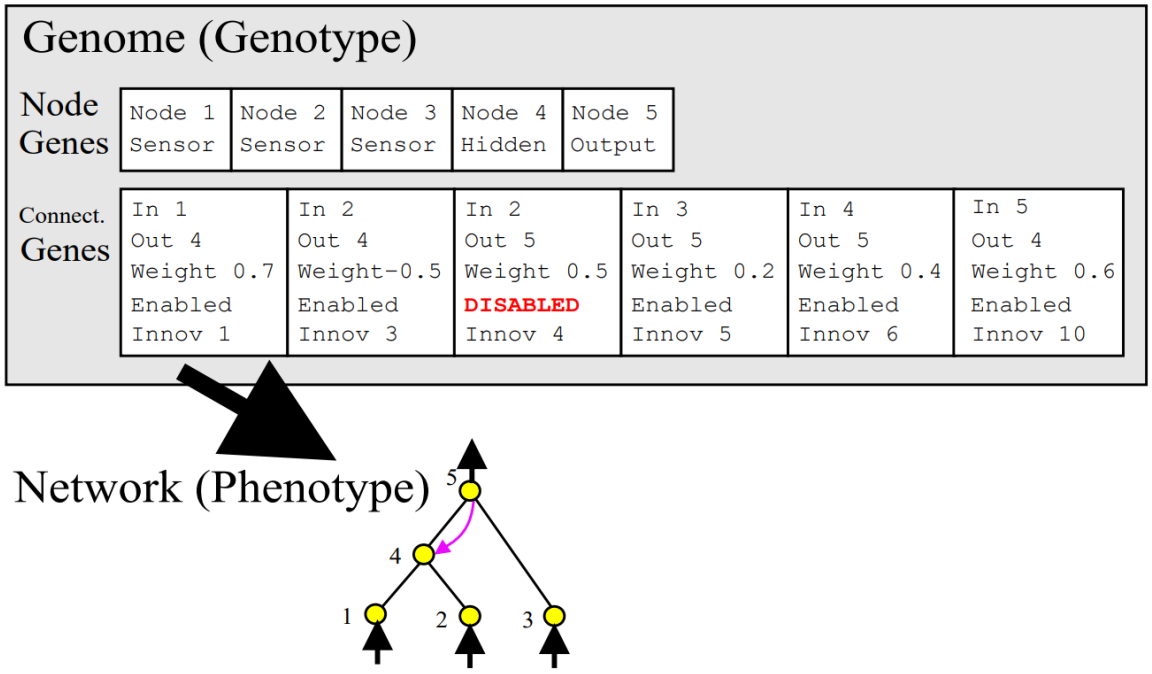


FIGURE 2.5: Representation of Genotype and Phenotype of NEAT individual [102]. Node genes display all nodes in the network. Connection genes store all information about the connection, such as in and out nodes, the weight of the connection, if it is enabled or disabled, and the innovation number.

There are two types of mutations that can happen, adding a node or mutating existing connections. If a node is added between two nodes (start node and end node), the previous connection is disabled but remains in the genome and two new connections are added: one from the start node to the newly added node with a weight of the previous connection that is disabled now, and the second connection from the newly added node to the end node with a weight of 1. If the connection is added between the start and end nodes, the weight will be initialized randomly. Weights will be perturbed in every generation. In this way, the neural network will be complexified gradually.

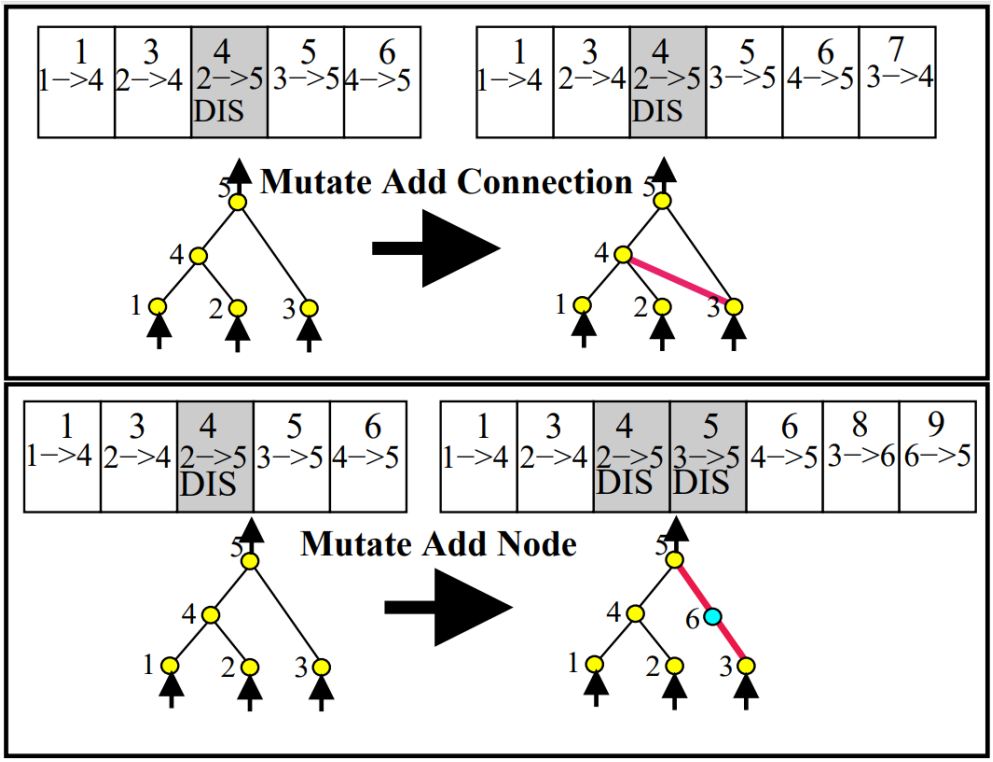


FIGURE 2.6: The diagram illustrates structural mutation with the addition of connection and nodes [102]. Connection genes represent the mutations. The genotype, which is above the phenotype (or structure of the network), starts with an innovation number. An innovation number is a historical marker that is used to identify the ancestors of the node. The genotype then contains a connection between nodes. In *add connection* mutation, we add a connection which then gets an incrementally new innovation number. For *add nodes* mutation, we first disable the connection $3- > 5$ then add a new node 6, which will add two new genes 8 and 9.

One of the major problems in NE was of *Permutations Problem* [82] or, more commonly known as, *Competing Convention Problem* [65]. Competing conventions means having more ways to express the solution to an optimization problem with NN. When genomes, that are solving the same problem but are not having the same structure, perform crossover, the resulting child will be damaged and will be non-functional. To address this, NEAT introduced historical marking, as we can see in Figure 2.7. Every gene in a genome is assigned a unique number. When two individuals are about to crossover the genes with similar numbers are aligned and the

crossover happens. The rest of the genes are called either disjoint or excess, depending on if they occur within the markings of similar genes or outside, respectively. These excess and disjoint genes are inherited from the more fit parent.

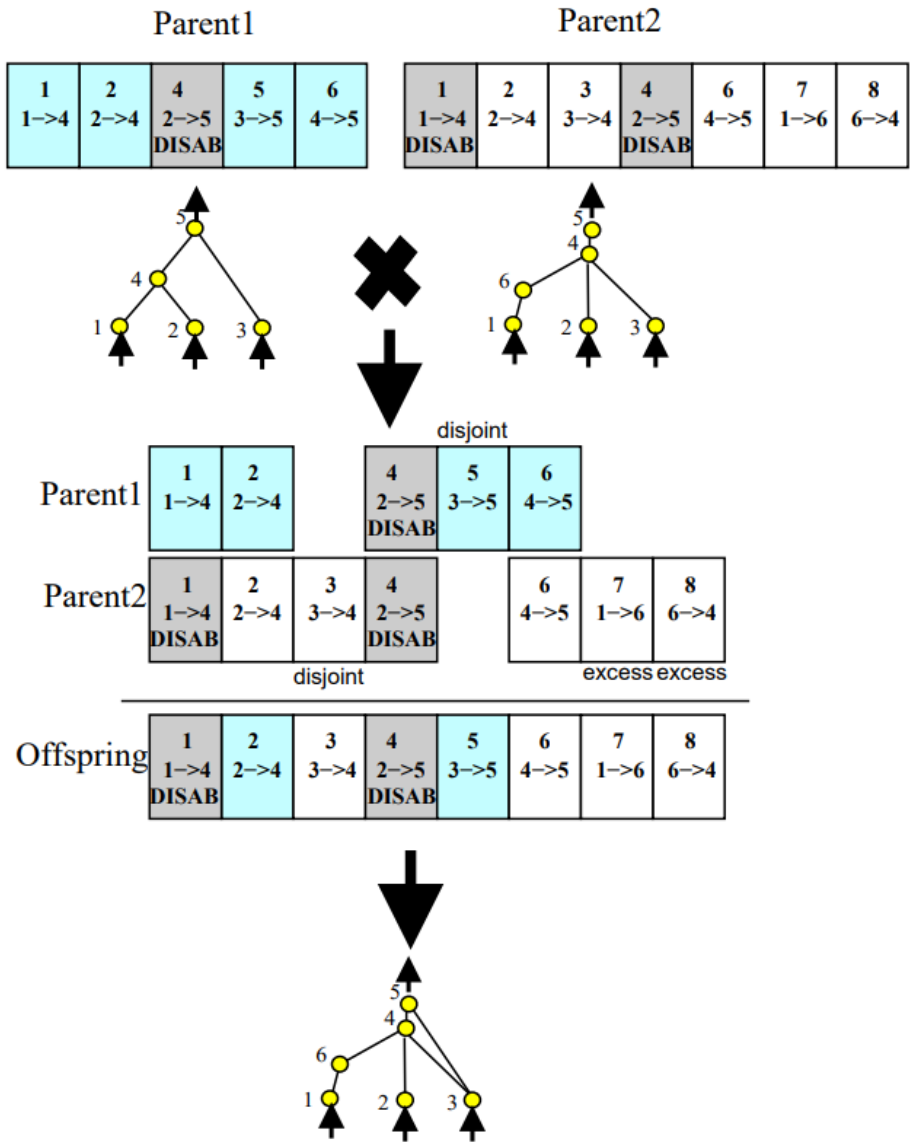


FIGURE 2.7: Figure displays crossover between two parent genomes and explains how similar, disjoint and excess genes work [102]. From similar genes, we will always select from the fitter parent. The disjoint and excess genes are always inherited as they are. Innovation number defines excess and disjoint genes. Disjoint means gene present within the range of maximum innovation number of the other parent and excess means out of the range of maximum innovation number present in the other parent genotype.

Speciation [2] was studied in Genetic Algorithms before it was applied to NE in NEAT for the first time. Speciation creates species in the population by measuring a *delta* coefficient which calculates how similar are the individuals to each other. Explicit fitness sharing then allows individuals with similar genomes to share fitness thus preserving innovation and creating diversity.

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3.W \tag{2.2}$$

In Equation 2.2, δ is the delta coefficient, $c1, c2$ and $c3$ are coefficients that give importance to each factor while N is the number of genes in the larger genome. E and D and number or excess and disjoint genes. W is the average weight difference of similar genes. δ allows us to speciate using compatibility threshold δ_t. $\delta < \delta_t$ means the genomes belong to the same species.

NEAT beats tough benchmarks like double pole balancing without velocity against state-of-the-art NE methods, like Cellular Encoding and Enforced Subpopulations. The authors also did a comprehensive ablation study to prove that all parts of NEAT are necessary to get the best results. NEAT has also demonstrated superior performance when compared to fixed topology approaches, and has been used in numerous subsequent research works to great success [14, 56, 90, 97, 100].

2.7 Open-ended Learning

Open-ended Learning is a field of Machine Learning where two populations of agents and environments co-evolve endlessly [99]. Theoretically, this algorithm should not stagnate and should produce artifacts endlessly. Though any Machine Learning algorithm can be used in an Open-ended Learning setting, Evolutionary algorithms are generally used to evolve both populations.

Co-evolution is a necessary part of an Open-ended Learning system. Co-evolution is an Evolutionary Algorithm that evolves more than one population together. The co-evolution is set up in a collaborative or competitive setting. In collaboration, it helps each other to achieve a common goal, while in a competitive setting, it competes with each other to achieve the goal.

For the Open-ended Learning algorithm, a cooperative co-evolution named Minimal Criterion Co-evolution (MCC) is used [7]. MCC is a co-evolutionary algorithm

that states that the problem can only evolve when the solution meets some minimum criterion. For instance, the research it was introduced was co-evolving agents and environments. The environment (problem) could only evolve when one of the agents (solution) from the population of agents could solve the environment and the newly created environment should be solved by one of the agents. This is also known as Problem-Solution Co-Evolution (PSCE) [22].

MCC led to the creation of theoretically true Open-ended Learning algorithms. By definition, an Open-ended Learning algorithm is one which can run endlessly while discovering stepping stones to novel artifacts without stagnating. The co-evolution of environments and agents is well suited for this definition, as the environment can be evolved endlessly if the agents can keep on solving them. Although any Machine Learning algorithm can be used in the Open-ended Learning paradigm, evolutionary algorithms are more commonly used and are best suited for it as inherently they have a high degree of exploration [3].

2.8 POET and Enhanced POET

To explain our research we first explore POET and Enhanced POET (EPOET) to make the understanding of the baseline clear. **POET** focuses on evolving pairs of agents and environments in an attempt to create specialist agents of that particular environment. POET experiments on a variant of the 2D Bipedal Walker Hardcore environment from OpenAI Gym [8] where it uses a subset of obstacles within the environment to create a child environment. The first environment is a basic plain surface environment and as generations of evolution progress, the environment complexifies. Table 2.1 explains all obstacles and how much they are able to mutate in a single generation. Figure 2.8 shows a possible terrain for the agent to solve.

TABLE 2.1: Environmental parameters for POET [115].

Obstacle Type	Stump Height	Gap Width	Step Numnber	Roughness
Initial Value	(0.0,0.4)	(0.0,0.8)	1	Uniform(0,0.6)
Mutation Step	0.2	0.4	1	Uniform(0,0.6)
Max Value	(5.0,5.0)	(10.0,10.0)	9	10.0

POET also transfers agents from one environment to another. Transfer of agents across environments is a crucial feature of POET as it not only helps with an agent

that might be stagnating in a local optimum but also helps in creating agents that can perform better to some extent, on other environments which leads us towards generalization of agents in POET. The transfer of an agent is attempted by first fine-tuning the candidate agent on the target environment and then comparing the fitness with the incumbent agent. If the fitness is higher than the incumbent agent, POET replaces the candidate agent with the incumbent agent.

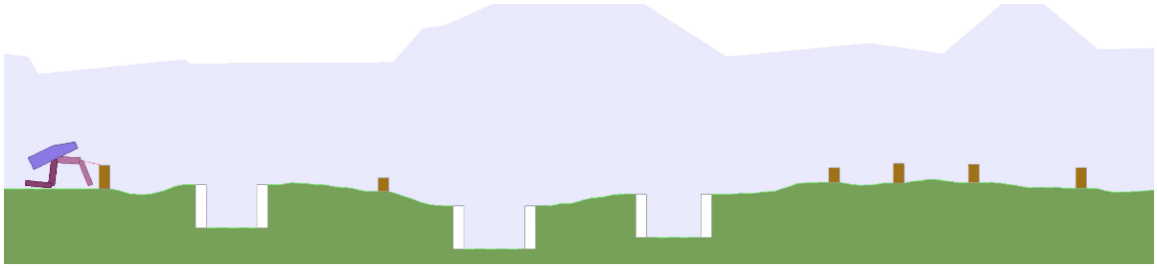


FIGURE 2.8: A view of the *2D Bipedal Walker Hardcore Environment* [115]. Possible obstacles are stumps, gaps, steps and roughness of the surface.

There is a preset reproduction eligibility condition for a child environment to be selected in the list of possible parents for a mutation to create the child environment for the next generation. This reproduction eligibility ensures that the agent in child environment has solved the environment. Once the children are selected, they are uniformly sampled to select parents to be mutated by *mutation step* in Table 2.1 of some or all of the available obstacles. Then the *Minimal Criterion* (MC) is applied to filter out environments that are too easy or difficult. The environment which passes MC and has the lowest *environments difficulty* score is selected. Thus, the three main goals of POET are to:

- generates new environments from active ones.
- optimize paired agents in respective environments.
- attempt to transfer agents from one environment to another.

EPOET brings 2 algorithmic innovations, 1 innovation for producing more expressed environments and 1 for generic measuring of the extent to which the system continues to show open-endedness. The first algorithmic innovation describes environmental encoding (EE), which is the mapping of the various parameters of

the environment that creates the environmental search space. Additionally, it describes environmental characterization (EC), which is the key attribute of the environment and helps in measuring the distance between environments. In POET, EC and EE are derived from the same set of static and hand-coded features that make it domain-dependent, limiting the search space and thus limiting the open-ended process. Thus, the first enhancement is to make EC generalized. Authors introduced *Performance of All Transferred Agents-EC* (PATA-EC) which is motivated by novel distinctions among the agents in the system. Four steps for PATA-EC are: (1) Evaluating the agents on all active and archived environments that have no extra computation as it was being calculated before for the transfer mechanism, (2) Clipping the evaluations to a maximum and minimum range as if the agent's high evaluation means it is already competent and low means it is an outright failure, (3) Rank these evaluations and (4) Normalize these rankings to -0.5 to 0.5 which allows the direct use of Euclidean-distance metric between PATA-ECs. PATA-EC revolutionizes POET as it calculates environmental novelty independent of the domain. Figure 2.9 illustrates the procedure to calculate the PATA-EC score.

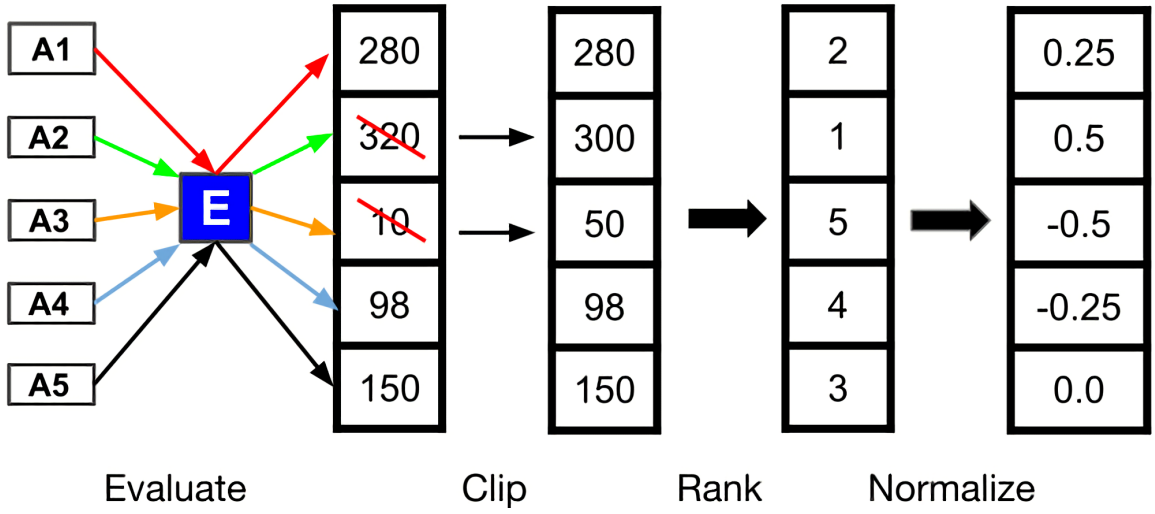


FIGURE 2.9: Figure illustrates the process to calculate PATA-EC score[114]. First, all active and archived agents are evaluated on the created environment *E*. The scores of the evaluations are clipped between a min-max range. The scores are ranked and then normalized. Thus, for one environment we get a vector of the length of the number of active and archived agents.

The second algorithmic innovation was in the *transfer mechanism* as POET's transfer mechanism created two issues: It was computationally expensive as all transfers needed to be fine-tuned and prone to false positives due to stochasticity in RL optimization. To counter these issues, authors introduced a more strict threshold that the candidate agent's (agent being transferred) score should exceed the incumbent agent's (agent being replaced) 5 previous scores which include both direct and fine-tuning scores. This smooths out noise in ES optimization and reduces computational cost as only those will be fine-tuned that have passed transfer test.

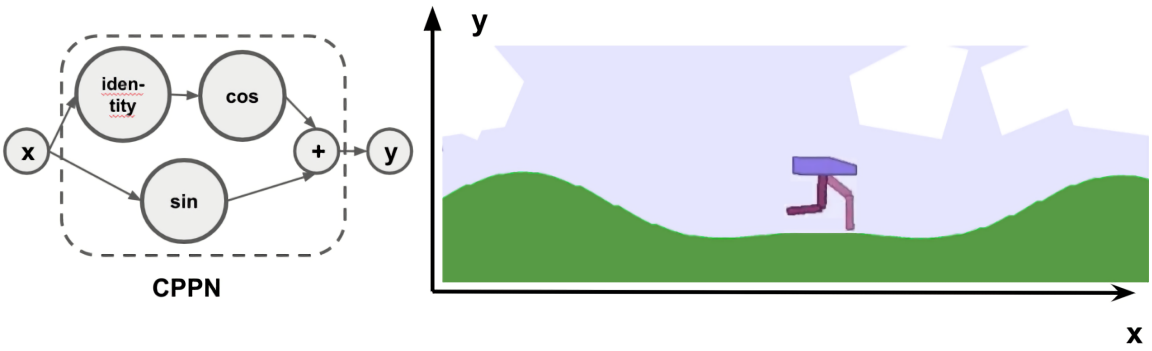


FIGURE 2.10: A CPPN generator (left) generating a level (right). [114]

The innovation for producing more expressive environments counters the issue in POET that it produced obstacles from the 2D Bipedal Walker Hardcore Environment and that is also a subset of it, as it limits the expressiveness of the environment generation algorithm. POET might produce many environments but at some point of complexity, the pressure of novelty will stagnate as there is no expressive algorithm that is generating environments. To tackle this, EPOET uses Compositional Pattern Producing Networks (CPPN) [97], a type of NN that takes in x coordinates as inputs and produces y coordinates as output, where x is the width of the terrain and y is the height. A CPPN uses many different types of activation functions that produce patterns. CPPNs are evolved through NEAT to make patterns that complexify procedurally as at first CPPN are initialized with basic architecture, i.e, only input and output neurons and gradually NEAT evolves and complexifies CPPN and thus complexifying the environment. Figure 2.10 shows a CPPN generator taking the x coordinate from a 2D environment, applying different activation functions, and producing the y coordinates. PATA-EC score then makes it possible to

measure diversity in the produced environments, thus the addition magnifies the open-endedness of POET.

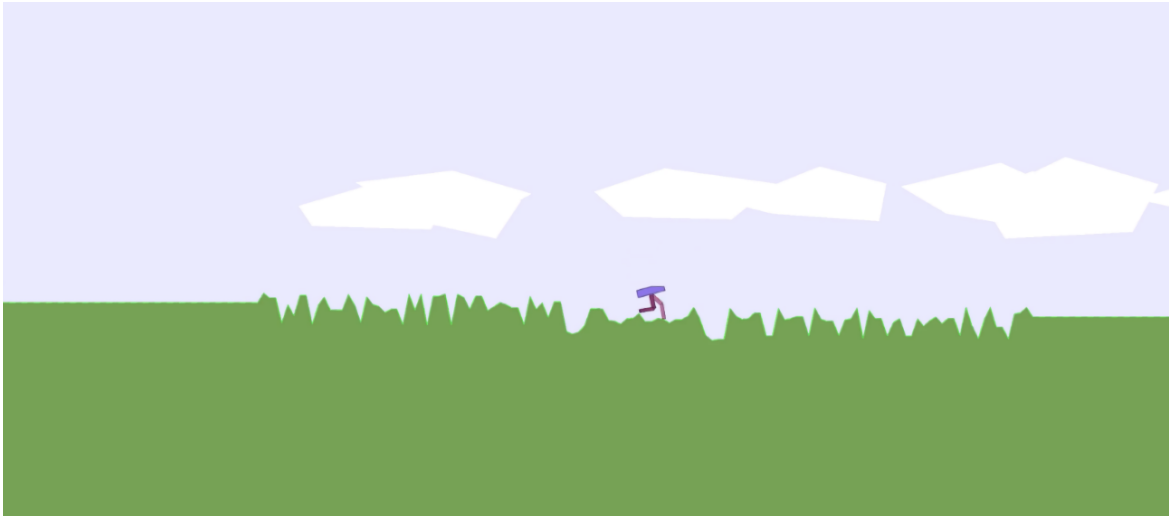


FIGURE 2.11: Example environment produced by EPOET. [114]

Lastly, the authors introduced *accumulated number of novel environments created and solved* (ANNECS), a metric for an Open-ended Learning system that takes into account two factors to evaluate an environment: (1) must pass the minimal criterion against all agents, active and archived, generated over the entire run so far, and, (2) must eventually be solved by the system, to prevent the system from receiving credit for generating harder environments. If ANNECS gets higher scores as the run proceeds indicate that the system is consistently generating meaningful new environments. Algorithm 2 explains the overview of all steps required by EPOET to produce environments open-endedly. Figure 2.11 shows the difficulty and expressiveness of an environment by having different micro and macro features.

High-level Algorithm 2: Overview of EPOET Algortithm

1. Create initial environment_agent (EA) pair with the flat environment and randomly initialized agent.
2. Optimize agent till it passes reproduction_threshold. Check if the EA_score lies in between minimal_criterion (mc).
3. Create environment through CPPN. Check novelty PATA-EC score.
4. Check if a transfer of agent is possible.
5. Archive oldest EA pair
6. Compute ANNECS.
7. Reiterate from step 2.

2.9 Related Work

POET has led to an explosion of new use cases such as PINSKY [19] was introduced which extends the POET algorithm for game level generation and agents that solved those levels for 2D Atari games. Quessy and Richardson [81] use unsupervised skill discovery [9, 28, 92] in the context of POET to discover a large repertoire of useful skills. Meier and Mujika [62] also investigated unsupervised skill discovery through reward functions learned by neural networks. Other uses of POET include the work by Zhou and Vanschoren [121], who obtained diverse skills in a 3D locomotion task. POET has also been shown to aid in evolving robot morphologies [104] and avoiding premature convergence which is often the result when using handcrafted curricula. Norstein, Ellefsen, and Glette [70] use MAP-Elites [68] to open-endedly create a structured repertoire of various terrains and virtual creatures. Ølberg [72] looks into Enhanced POET and tries to find difficulties in the generated environment to reduce the computational cost.

Adversarial approaches are commonly adopted when developing open-ended algorithms. Dennis et al. [18] propose PAIRED, a learning algorithm where an adversary would produce an environment based on the difference between the performance of an antagonist and a protagonist agent. Domain randomization [88], prioritized level replay [44] and Adversarially Compounding Complexity by Editing Levels (ACCEL) [78] adopt a similar adversarial approach, where teacher agents

produce environments and student agents solve them.

Several domains and benchmarks have been proposed with the aim of encouraging research into open-ended, general agents. Team et al. [110] introduce the XLand environment, where a single agent is trained on $700k$ 3D games, including single and multi-agent games, resulting in zero-shot generalization on hold-out test environments. Later on, Team et al. [109] used an attention-based memory architecture [17] to show an agent's emergent capabilities on held-out tasks while training open-endedly. Barthet, Liapis, and Yannakakis [4] introduced an autoencoder [12, 54] and CPPN-NEAT based open-ended evolutionary algorithm to evolve Minecraft [13, 55] buildings. They showed how differences in the training of the autoencoders can affect the evolution and generated structures. Fan et al. [29] create a Minecraft-based environment, MineDojo, which has numerous open-ended tasks. They also introduced MineCLIP as an effective language-conditioned reward function that plays the role of an automatic metric for generation tasks. Kepes, Guttenberg, and Soros [49] introduces ChemGrid, a benchmark for general game-playing agents based on artificial chemistry. The agents in ChemGrid can discover new molecular recipes by making or breaking existing recipes. Gan et al. [34] introduce the Open-ended Physics Environment (OPEn) to test learning representations and tested many RL-based agents. Their results indicate that agents that make use of unsupervised contrastive representation learning, and impact-driven learning for exploration, achieve the best result.

Another related field is that of lifelong, or continual learning. Here, there is often only one agent, which is in an environment with multiple tasks [51, 57]. The agent can then continuously improve as it experiences new settings. A challenge here is how to transfer knowledge between different tasks while preventing *catastrophic forgetting*, i.e. where an agent performs worse on previously learned tasks after fine-tuning on new ones [74, 77]. In particular, Rusu et al. [87] introduce *Progressive Neural Networks* where, for each new task, the existing network is frozen and extra capacity is added, which can learn in this new task. This allows the model to leverage lateral connections and transfer information from previous tasks, while not diminishing the performance on previously learned tasks. Progressive NNs do not maintain a population of networks thus they only focus on increasing the capacity of the network and not optimizing it. Other works attempt to keep the agent actions unchanged on these old tasks. Oswald et al. [74] do this by learning a model

that transforms a task identifier into a neural network. For new tasks, the loss incentivizes the model to output similar weights for already learned task identifiers. Li and Hoiem [59] use a similar technique, where prior outputs from the model should not change significantly when learning a new task.

We observe that related works have been implemented through a fixed topology Neural Network (NN). We hypothesize that agents that can augment their NNs will have more capacity to solve complex environments that are created in the stage of an Open-ended Learning system. We also hypothesize that these agents will have more generalization capabilities through different transfer mechanisms. Therefore, we propose to extend Enhanced POET agents through NEAT which will be explained in chapter 3.

Chapter 3

Research Methodology

This research aims to improve Enhanced Paired Open-Ended Trailblazer (EPOET) algorithm by introducing the Neuroevolution of Augmenting Topology (NEAT) algorithm to continuously evolve the Neural Network's (NN's) topology so that we may have a true Open-ended Learning algorithm. This is motivated by EPOET's fixed topology agents. This chapter will explain, in detail, what methods we used to achieve our goals. We will start off by explaining the existing algorithms we used in our experimentation, followed by an explanation of our methods.

3.1 Open-endedly evolving the topology of agents

Many of the approaches introduced in prior work have been implemented using fixed topology approaches which motivates us to explore NEAT and the benefits it brings to the Open-ended Learning framework. We first describe the whole process of our proposed method, Augmentative Topology Enhanced POET (ATEP) in subsection 3.1.1, which will be followed by our experimental setup in subsection 3.1.2 where we will describe setting up each component of our algorithm.

3.1.1 Augmentative Topology Enhanced POET

In this section, we discuss all the building blocks of our algorithm and the different variants we experimented with. ATEP combines EPOET with NEAT to allow the agents' network topologies to evolve. This means that the algorithmic steps are very similar to EPOET, which we will be looking at, and the main differences are (1) the optimizer used: we use NEAT to optimize the variable-topology agents whereas EPOET used Evolution Strategies to optimize fixed-topology agents; and (2) the

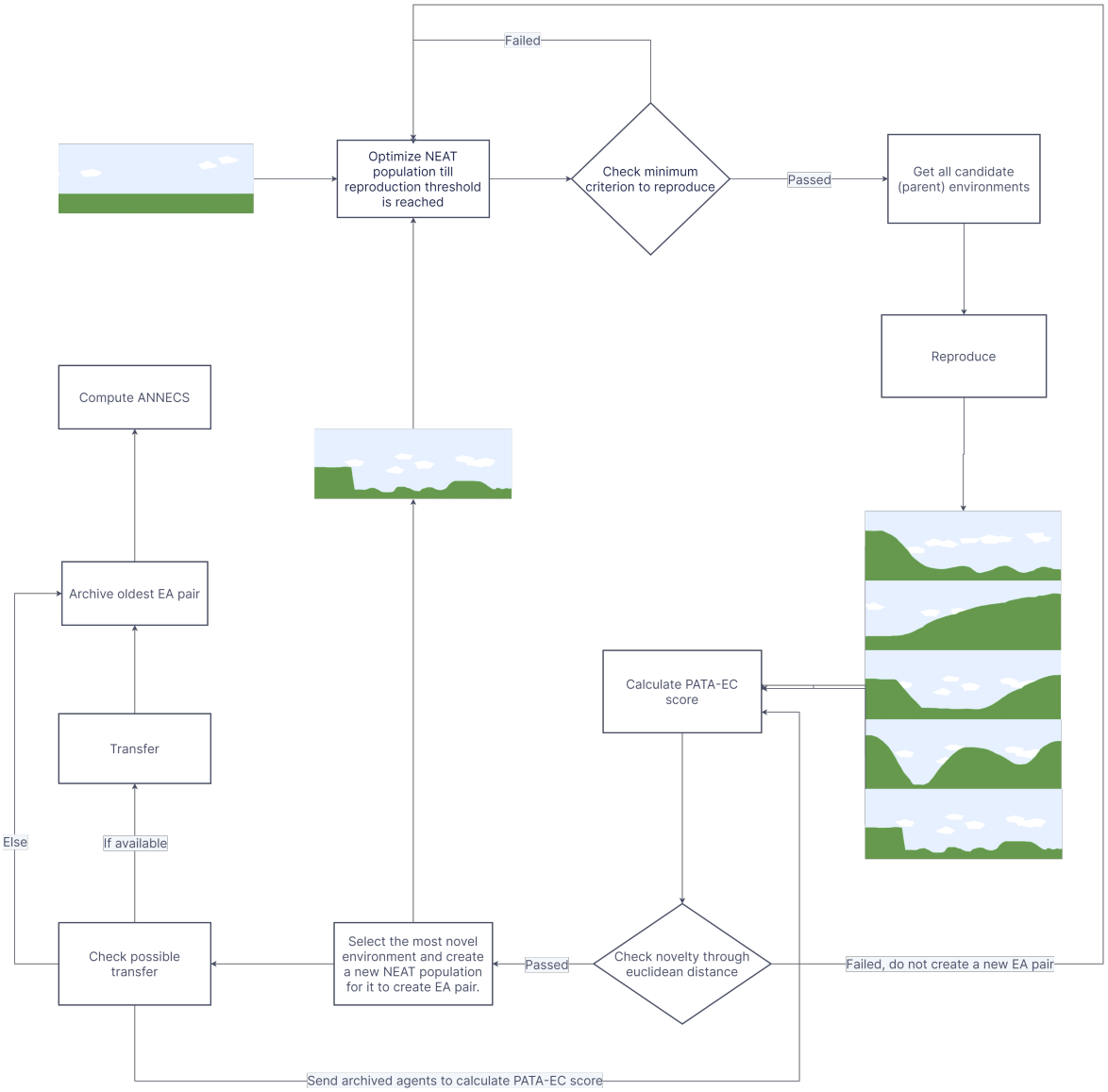


FIGURE 3.1: A flowchart demonstrating the flow of the ATEP framework, with blocks in green being where ATEP differs from POET. For both EPOET and ATEP, each environment is associated with an agent, represented by an ES population for EPOET and a NEAT population for ATEP. The environment images used in the chart were created by ATEP. Refer to Appendix A for pseudocode describing the transfer mechanisms used in ATEP.

transfer mechanism, which will be discussed later in this section. The detailed flow of ATEP that we will be talking about is also illustrated in Figure 3.1.

We start off with a simple environment and we initialize a NEAT population of agents with it. This will be our first Environment-Agent (EA) pair. We will then optimize the NEAT population to solve the environment, once a preset reward threshold is crossed, we check the minimum criterion to reproduce a child environment. This minimum criterion checks if the environment was too easy or too hard for the best agent in the population, which is checked by the reward it attains. As we get more active environments, this minimal criterion filters out only those that are likely to produce children that will not be unsolvable. If the minimal criterion is not passed we optimize the NEAT population to solve the environment untill it does meet the criterion. We then use CPPNs to produce a set of children. CPPNs are evolved by NEAT as well so the generation of environments gets procedurally difficult. Minimal criterion is also checked on these child environments and if none of them pass the criterion, no environment is created in the iteration.

These child environments then undergo the PATA-EC score process and then the Euclidean distance is applied to the calculated PATA-EC score to find the most novel environment which is selected as the new environment and a new NEAT population is created with it. This environment goes to the optimization pipeline. If the number of active environments is exceeded by a set limit, we archive the oldest and compute the ANNECS score. For computing the ANNECS score, we see if the paired populations' best genome exceeds the reproduction threshold, we consider it as solved and add one to the ANNECS score. After every N iteration, we try to transfer the population from one environment to another. We have created two types of transfer mechanisms, Fitness-Based Transfer and Species-Based Transfer.

Fitness-Based Transfer

Fitness-Based Transfer-ATEP (FBT-ATEP) is motivated by the original EPOET transfer mechanism and adjusted to integrate NEAT in it. We first pick up a candidate population, the one which is transferring, and look at the best genome. The best genome G_C is already being calculated for NEAT's evolution. We then take the target population's 5 previous best genomes and compare them with the G_C. If the G_C is better than all 5 previous best genomes of the target population, we fine-tune G_C on the target environment for M generations and then we compare it with the best score of the previous 5 target population's best genomes. If it is better, we transfer

the whole candidate population and replace it with the target population. Algorithm 3 shows the pseudocode for FBT-ATEP.

Algorithm 3: Fitness-Based Transfer

Input : Candidate population's best individual I, a function *Score(.)* that calculates the maximum of the target agent's 5 most recent fitness scores.

Let M = All environments - {Candidate environment}
foreach m *in* M **do**
  Compute direct transfer I_D;
  if $I_D > Score(m)$ **then**
    Compute fine-tuning transfer I_P;
    if $I_P > Score(m)$ **then**
      Add m to $T_candidates$
    else
      Transfer not possible
    end
  else
    Transfer not possible
  end
end
Delete the whole population of $T_candidates$
Transfer whole candidate population to $T_candidates$

Species-Based Transfer

For the second transfer mechanism, we use the speciation inherent in NEAT to influence transfer. Specifically, we check if the best genome in the candidate population is within a δ threshold (using the speciation calculation in Equation 2.2) of any target environment's best genome, where the best genome is already being calculated for NEAT evolution. If this is the case, we transfer the candidate species and replace the target species with it. This approach, called *Species-Based Transfer ATEP* (SBT-ATEP), skips the step of comparing fitness scores and has its own advantages which we discuss in the next section. Algorithm 4 shows the pseudocode for SBT-ATEP. Finally, we also consider random transfer (RT-ATEP) and no transfer (NT-ATEP) to investigate whether the transfer mechanisms have a large impact on

the results.

Algorithm 4: Species-Based Transfer

Input : Candidate population's best individual I_c. A function $find_delta(.)$ that calculates delta score and $\delta_{threshold}$.

Let M = All environments - {Candidate environment}

foreach $m \in M$ **do**

  I_m = best individual of environment m

  δ_{ct} = find_delta(I_c, I_m) using Equation 2.2

  if $\delta_{ct} \leq \delta_{threshold}$ **then**

    delete target species

    Transfer candidate species to target population

  else

    The transfer is not possible

  end

end

3.1.2 Experimental setup

Environments

Now we describe the experimental setup for ATEP, its variants, and our baselines. POET and EPOET have exhaustively shown that they are capable of producing extremely challenging environments [114, 115] thus the experiment setup is in order to solve complex environments with the least possible stagnation. We use a variant of OpenAI Gym's [8] *2D Bipedal Walker Hardcore* environment as it is being used by EPOET. The reason provided by Enhanced POET's authors to use this environment is the simplicity as a 2D walking domain with the possibility of creating many terrains. Thus, this motivates us to select this environment. As opted by EPOET, we will also be considering ground roughness as the only parameter to mutate as it creates much diversity and, again, extremely challenging environments. CPPNs are used to mutate ground roughness parameters and are evolved through NEAT. A node may have one of the following as an activation function: identity, sin, sigmoid, square, and tanh. We default the activation function to identity. All our variants and baselines have the same CPPN hyperparameter settings. Refer to Table A.4 for detailed hyperparameter settings.

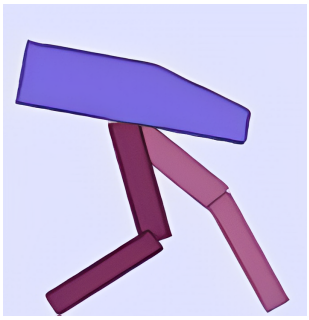


FIGURE 3.2: A two-legged agent with a hull from OpenAI Gym's 2D Bipedal Walker environments [8].

Agent

The agent's body has a hull that is supported by 2 legs, as illustrated in Figure 3.2. 2 motor joints control the hips and knees of each leg which creates a 4 dimension action space. The agent has 10 LIDAR rangefinders for perceiving terrain and the measurements from the LIDARs are included in the state space. Another 14 state variables including the position of joints and joint angular velocity, horizontal and vertical speeds, hull angle, hull angular velocity, and whether legs touch the ground, are included to make 24 dimension observation space [8].

The agent is required to navigate, from left to right within a time limit and without falling over, on an environment of generated terrain of different ground roughness parameters. As described in the Equation 3.1, the reward is given per step for moving forward. We set no constraints on agents' behaviour other than that the agent needs to keep its hull straight and minimize the torque.

$$Reward = \begin{cases} -100, & \text{if agent falls} \\ 130 * \Delta x - 5 * \Delta \text{hull_angle} - 0.00035 * \text{applied_torque}, & \text{otherwise} \end{cases} \tag{3.1}$$

The time limit for the episode to terminate is 2000 time steps. The episode terminates if either this time limit is reached, agent falls, or completes the course. In our experiments, being solved means the agent reaches the far end of the environment and obtains a minimum score of 200. In our observations, backed with EPOET experiments, 200 rewards are enough to ensure that the agent is efficient.

We use NEAT as the algorithm to augment the topology of our agents. We have a population size of 1000, which is sufficient to produce good results as per our experiments. We start off with a simple structure of 24 inputs (observation space) and 4 outputs (action space) for each individual. All nodes have tanh as the activation function. We crossover individuals with a probability of 0.3 of performing crossover. We mutate connections with a probability of 0.85 and add nodes with a probability of 0.15. For detailed hyperparameter settings for NEAT, refer to Table A.3. Settings for all our variants remain the same.

EPOET Framework

For the EPOET framework, we check if the transfer is possible every 25 iterations. We try to create a new environment every 150 iterations. The reproduction threshold is set to 200 rewards and the minimum criterion is set to be in a range of 50 to 340 rewards. We archive environment-agent pairs only if we have 20 active environments. All these settings are fixed across all ATEP variants and EPOET baselines. Refer to Table A.2 for detailed hyperparameter settings for the general EPOET framework.

ATEP Variants and Baselines

For our baselines, first, we take the original controller, which has 2 hidden layers, 40 nodes each, and we call it EPOET40x40. For the second baseline, we shrink the original controller to have 20 nodes in each layer and call it EPOET20x20. EPOET20x20 is created to investigate our hypothesis on different levels of fixed topology agents, where we state that a fixed topology agent will cease to learn at some point of complexity, thus different controller sizes of fixed topologies will help us to prove our hypothesis. Refer to Table A.2 for detailed hyperparameter settings for Evolutionary Strategies for our baseline controllers. For our variants, we have Species-Based Transfer ATEP and Fitness-Based Transfer ATEP (SBT- and FBT-ATEP), which are discussed earlier, and we add two more variants which are also based on transfer mechanisms, named Random Transfer ATEP (RT-ATEP) and No Transfer ATEP (NT-ATEP) which are added to show that transfer mechanism is an important part of Open-Ended algorithm. For RT-ATEP, if we have two EA pairs we transfer in both of the EAs and if we have more, we randomly take 3 EA pairs and transfer them to 3 random EA pairs and replace the population.

Hyperparameter Selection

All the hyperparameters belonging to EPOET were inherited from EPOET research, except the number of active environments which was reduced to 5 because of computational restrictions. All NEAT-related hyperparameters were selected after extensive experimentation and the best-performing hyperparameters were selected.

Chapter 4

Results and Discussion

In this chapter, we discuss and analyze our results. We break the results into 3 different categories: open-endedness, nodes complexity exploration, and generalization ability. All results are gathered based on 5 seeds due to the expensive computational load, with each algorithm requiring approximately 50,000 to 200,000 CPU hours for a single run. EPOET20x20 required the least amount of computation, while SBT-ATEP required the most. Each algorithm was run in parallel on a cluster consisting of 264 Intel Xeon cores, with the runtime ranging between 10 and 30 days. Each algorithm had a maximum of a 10^{12} function evaluation per run. One function evaluation corresponds to a single episode of at most 2000 timesteps long.

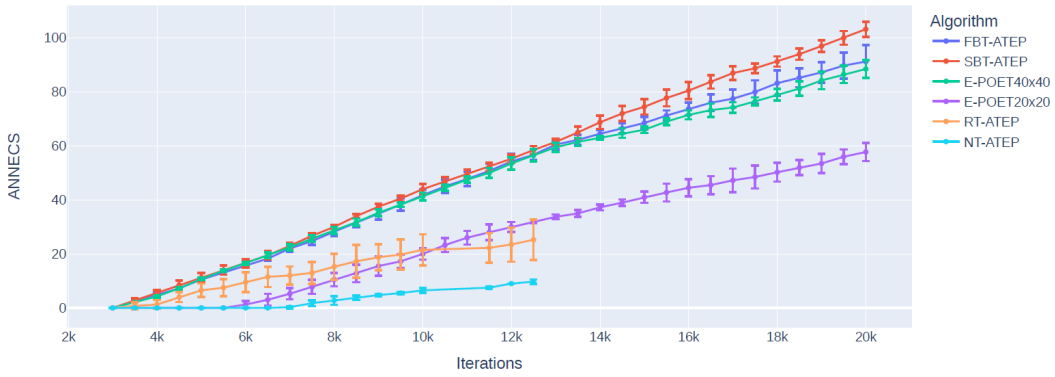


FIGURE 4.1: Accumulated Number of Novel Environments Created and Solved (ANNECS). To save the compute, we stop the NT- and RT-ATEP experiments early, as it is clear that they perform poorly. The error bars are standard deviation, the line represents the mean and both are calculated over 5 runs.

4.1 Open-endedness

As mentioned earlier, Wang et al. [114] introduced the ANNECS metric to capture the open-endedness of an algorithm; we take it as our most important score to judge which algorithm performs better on complex environments.

Figure 4.1 shows the ANNECS score as a function of training time. We see that there is a significant difference between EPOET20x20 and FBT- and SBT-ATEP, indicating that the small network results in solving fewer environments. EPOET40x40 performs substantially better than EPOET20x20 and is competitive with ATEP early on during training. The rate of increase in ANNECS, however, does decrease after about 13k iterations, whereas ATEP increases at a consistent rate. This substantiates our hypothesis that fixed topology agents will start stagnating at some level of environmental complexity, due to capacity issues. While we can improve the results by increasing the size of the network, that will merely delay the onset of stagnation.

FBT-ATEP outperforms EPOET40x40, although it also slows down slightly as time progresses. This is due to replacing the entire target population with the transferred population, which may eliminate all useful skills learned by the target population. SBT-ATEP, on the other hand, only replaces a single species that is close to the candidate species, leaving the rest of the population intact. We also find that SBT-ATEP has negligible delays in solving environments and, even though it performed similarly to FBT-ATEP and EPOET40x40 early on during training, it starts to outperform these in the second half of the experiment. This, as we will show later, is partly due to SBT-ATEP exploring more actions. We further note that the variations using no transfer (NT-ATEP) or random transfer (RT-ATEP) perform poorly, indicating that intelligent transfer mechanisms are necessary.

Although ATEP outperforms EPOET, it is more computationally expensive, as measured by the number of function evaluations. One function evaluation means one individual being evaluated on an environment. SBT-ATEP has the most function evaluations since once a species transfers from one population to another, it becomes highly probable that it can transfer in the opposite direction because it may now be within the $\delta_{threshold}$ range. This increases the population size, resulting in more function evaluations. The tradeoff here is of function evaluations to performance, which is justified as the performance confirms our hypothesis. Figure 4.2 displays the total number of function evaluations.

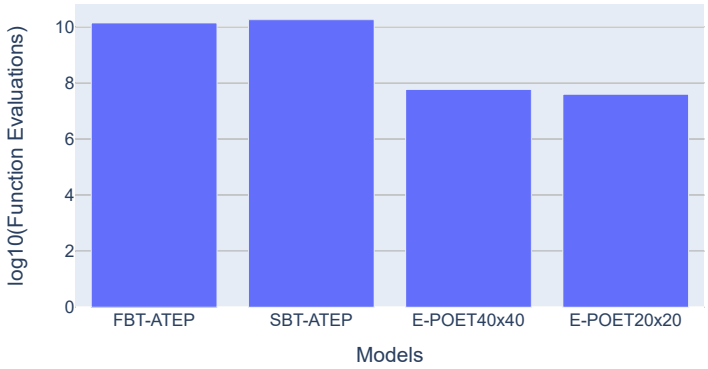


FIGURE 4.2: Cumulative sum of the number of function evaluations, with the Y-axis, converted to a log-scale. While ATEP requires significantly more function evaluations, we find that its total wall-time is only 3 times more than EPOET, as the neural networks are generally smaller and each evaluation does not take as long.

4.2 Nodes complexity exploration

We have now shown that SBT-ATEP outperforms all of the other tested methods based on the ANNECS score. We also find that it generally uses a smaller neural network with fewer nodes than the other algorithms. Figure 4.3 shows the number of nodes and corresponding fitness value for each algorithm. We can see that SBT-ATEP generally has a high fitness, but fewer nodes than the other approaches. This is echoed in Figure 4.4c, where SBT-ATEP has the least number of nodes for most of the experiment, although it gradually adds nodes and complexity. FBT-ATEP, on the other hand, adds nodes very rapidly. This again indicates that the transfer mechanism in EPOET is critical.

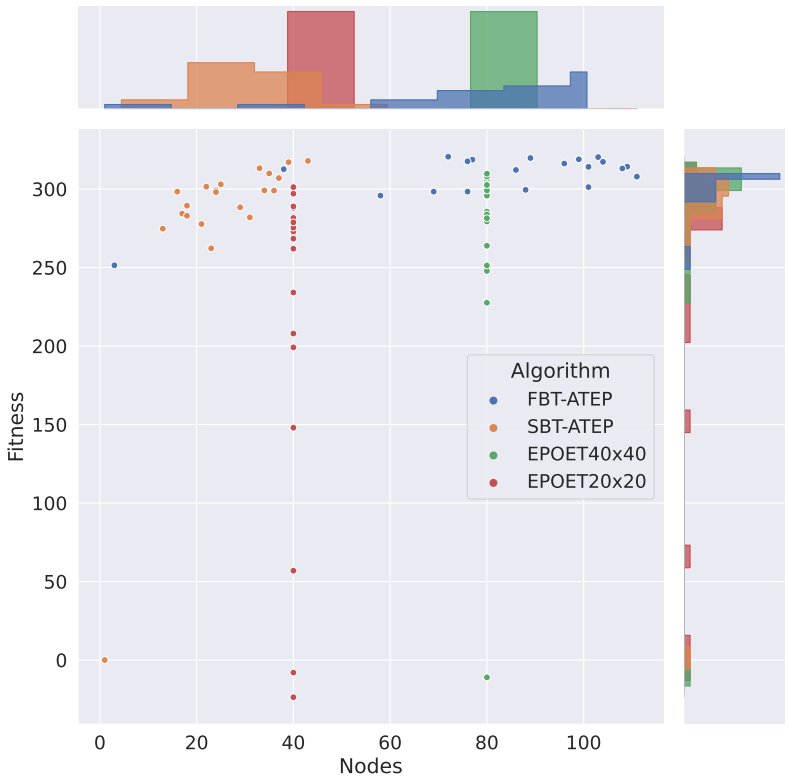


FIGURE 4.3: Mapping of fitness to nodes. We plot values every 1000 iterations, starting at iteration number 150. Each dot represents a specific iteration, as well as the mean fitness overall environments and the mean number of nodes in the population. The distribution on top represents the distribution of nodes while the distribution on the left represents fitness distribution.

Inspired by this, we further look into a simple *Fitness to Nodes* ratio (FNR) metric, shown in Figure 4.4a, and find that SBT-ATEP outperforms all other algorithms on this metric for the majority of the run. This indicates that SBT-ATEP outperforms all other algorithms on a per-environment basis while using fewer nodes. This leads us to believe that a better-curated transfer mechanism, based on SBT-ATEP, will sustain the FNR for longer runs.

Furthermore, in Figure 4.4b, we calculate an *ANNECS to Nodes* ratio (ANR) metric with the intent to observe the role of nodes in the open-endedness of the agents, i.e. to have the ability to complexify over time. We observe that SBT-ATEP performs significantly better than the other models. FBT-ATEP has the lowest ANR, as it adds nodes much faster than the rate of increase in ANNECS.

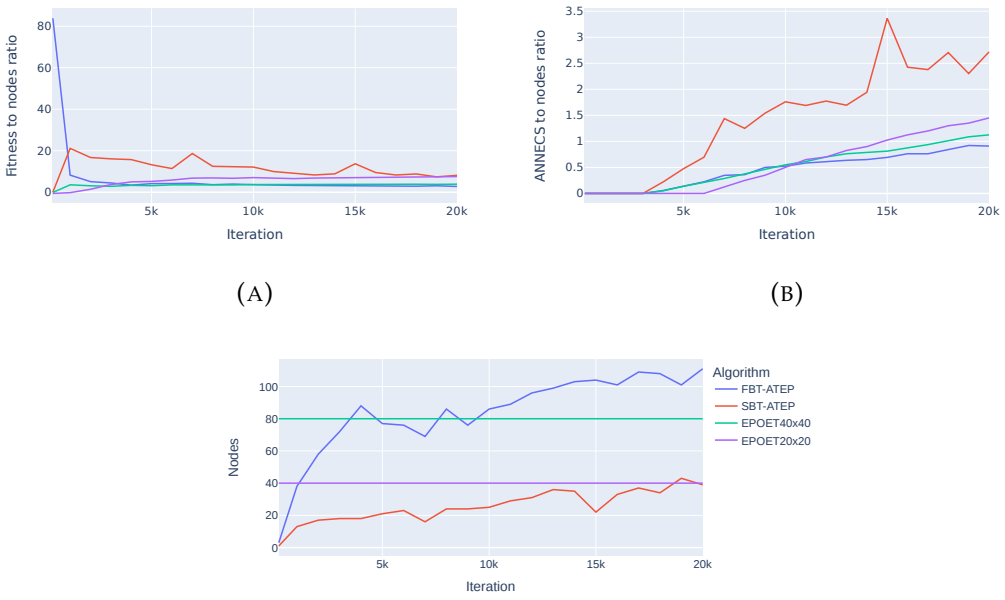
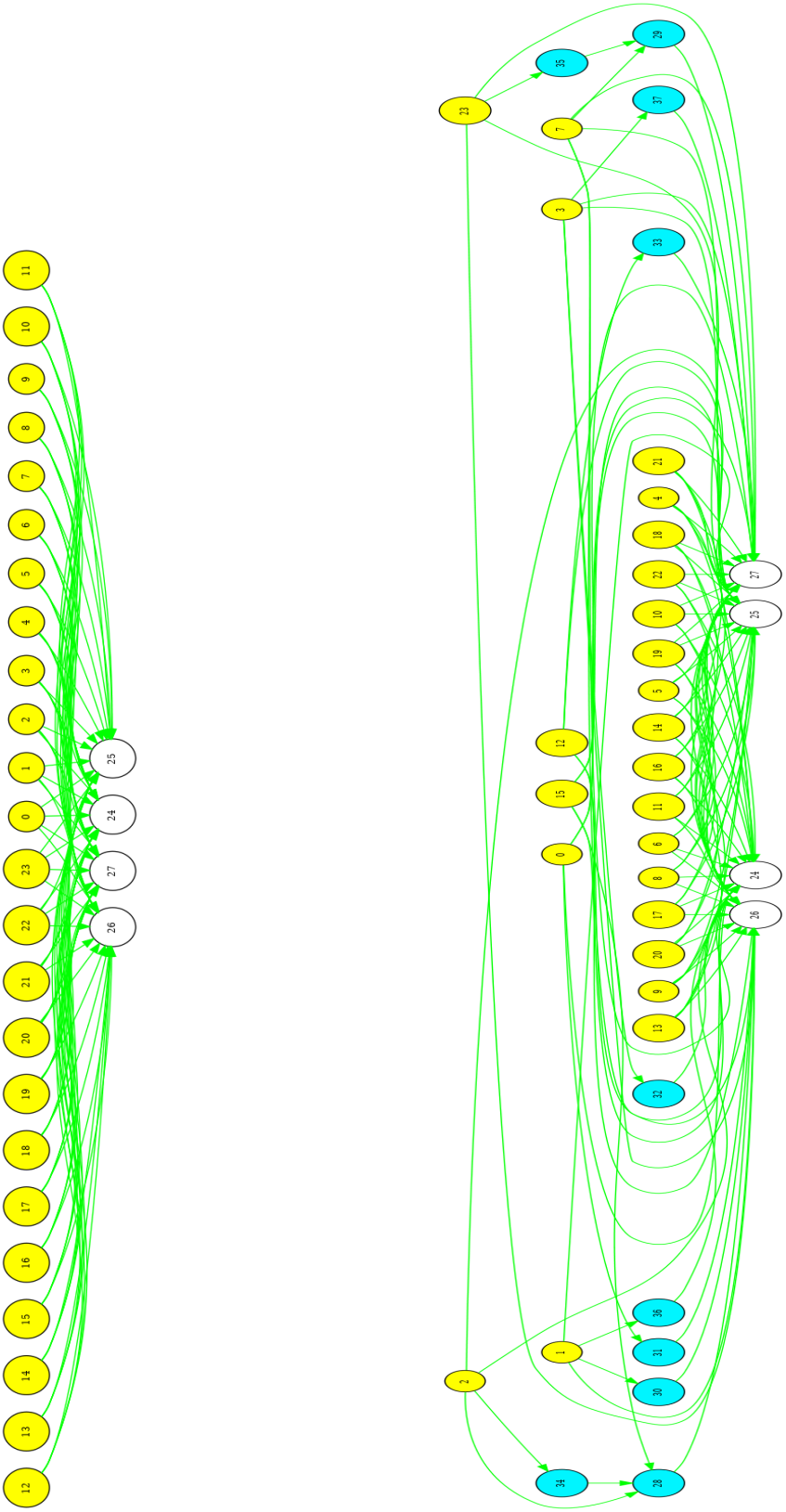


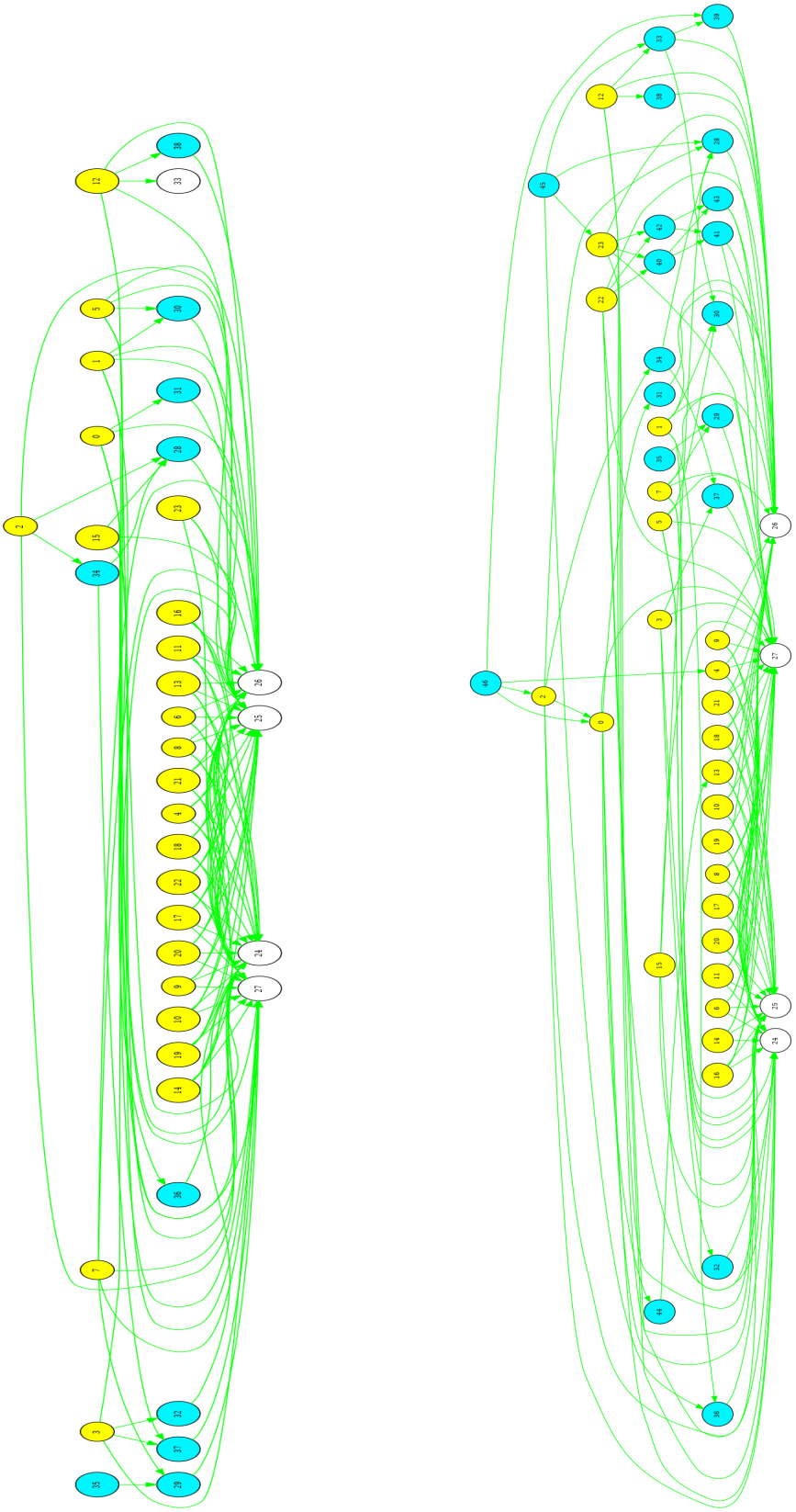
FIGURE 4.4: Analysis with respect to the number of nodes. Figure **(a)** shows FNR along iterations, **(b)** shows ANR along iterations, **(c)** shows the addition of nodes along iterations

Lastly, we explore how the evolved network's complexity emerges by looking into the network itself. NNs are illustrated in Figures 4.5 and 4.6 is the evolution of the best individual from a later stage environment. Yellow nodes represent inputs, aqua represents hidden nodes, and white maps to outputs. Figure 4.5a shows the start of the NN without any evolution operator applied. Figure 4.5b is the state after 2000 iterations. It shows to have a few nodes in hidden layer one and a couple of nodes in hidden layer two. Complexity has not yet emerged as can be observed. Figure 4.6a is the NN after 4000 iterations. We can observe emerging complexity. Some residual connections can be seen, for example, node 5, an input node connects to hidden node 30, but also connects to an output node 26. Figure 4.6b is the evolved network that became the best individual in the population, after 6000 iterations of when it was created. Here complexity is observed in many levels. Firstly, input node 2 connects to input node 0, whereas node 46 appears to be having a forward pass to input 2 and 0. Secondly, we observe an interesting small block of 6 nodes conventionally fully connected, input 22 and 23 connected to 40 and 42 which further connects to 41 and 43, which connects to the outputs. The overall depth of the network seems to be variable.



(A) (B)

FIGURE 4.5: From left to right: NNs at iterations 0 and 2000 of when it was created.

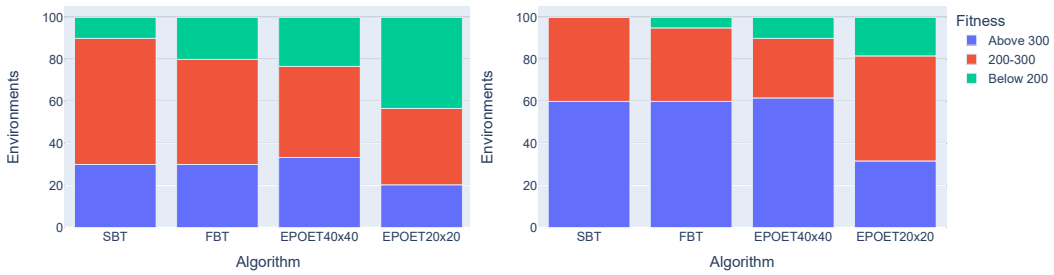


(A) (B)

FIGURE 4.6: From left to right: NNs at iteration 4000 and 6000 of when it was created.

4.3 Generalization Ability

We next evaluate the generalization ability of our open-ended agents, as Team et al. [110] has shown that these agents have the potential to generalize to new unseen environments. To concretely test this, we first take the 20 latest environments from each method. For each environment, we take the latest agent that could solve this environment from the method under consideration. Each of these agents is now evaluated on the selected environments from the other methods (60 in total). We perform 30 runs per environment-agent pair and calculate the mean and maximum of the rewards. We split the results into three categories: environments with fitness scores above 300, between 200 and 300, and below 200. Scores below 200 indicate that the environment has not been solved by the agent. Figure 4.7b shows the performance of each method when evaluated on the 60 other environments. We observe that SBT-ATEP outperforms all other models, with only 10% of the environments remaining unsolved when we take the mean score. When we look at the max score, we observe SBT-ATEP solves all environments at least once in the 30 runs we look at. This may be the stochastic behavior but it also indicates towards possibilities of powerful generalization capabilities if we have the right transfer mechanism.

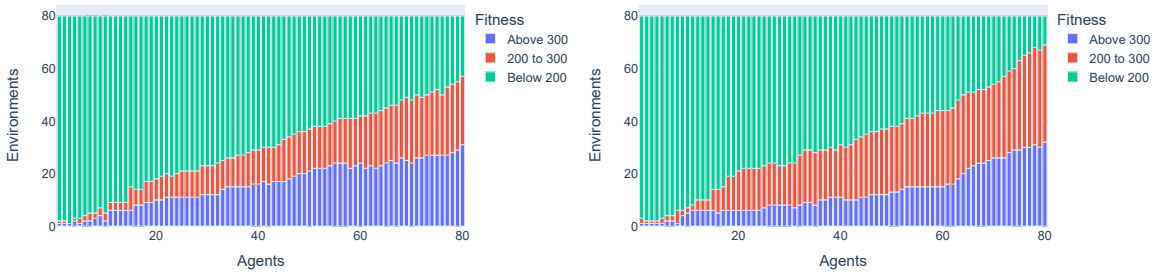


(A) (B)

FIGURE 4.7: Figure **(A)** shows each algorithm being tested on the 20 latest environments created by all other algorithms, i.e., each algorithm is evaluated on 60 environments. The Y-axis shows the percentage of environments in each category. Each test is conducted for 30 runs and the mean scores are taken. Figure **(B)** shows the maximum score of the 30 runs.

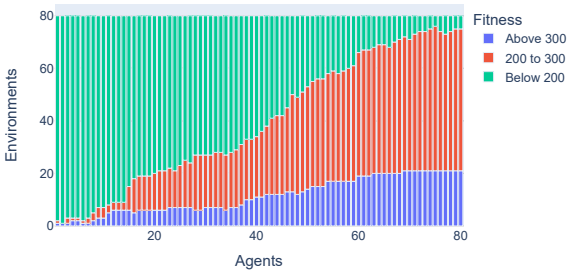
Secondly, we test the generalization capabilities of agents on all of the environments created by their own algorithms. We exclude EPOET20x20 as it fails to solve 80 environments in the whole run. We take into account 80 environments that were

solved by the model itself and observe how each agent performs on all of them. Figures 4.8a, 4.8b and 4.8c show the results. Here, early-stage agents perform worse and late-stage agents are shown to have generalization abilities on previously unseen landscapes. The transfer mechanism plays a key role in this generalization, as it exposes agents to more environments. Despite not having seen all environments, late-stage agents generalize much better. SBT-ATEP generalizes the best, with the lowest proportion of unsolved environments, in contrast to the lower-performing EPOET40x40 and FBT-ATEP.



(A) EPOET40x40

(B) FBT-ATEP



(C) SBT-ATEP

FIGURE 4.8: Figures showing generalization capabilities. Figures **(a)**, **(b)** and **(c)** show agents of 80 solved environments being tested on all 80 environments, for EPOET40x40, FBT-ATEP, and SBT-ATEP respectively. Note that EPOET20x20 does not take part in this test as it failed to produce 80 environments in the run.

Finally, we briefly investigate potential reasons why SBT-ATEP outperforms FBT-ATEP and EPOET40x40. We find that SBT-ATEP explores more actions, as it only transfers a single species instead of replacing the target population as is done by FBT-ATEP. This allows the new species to complement the actions that were already explored by the existing population. Figure 4.9 shows the action distribution

of each action for SBT-ATEP, FBT-ATEP, and EPOET40x40.

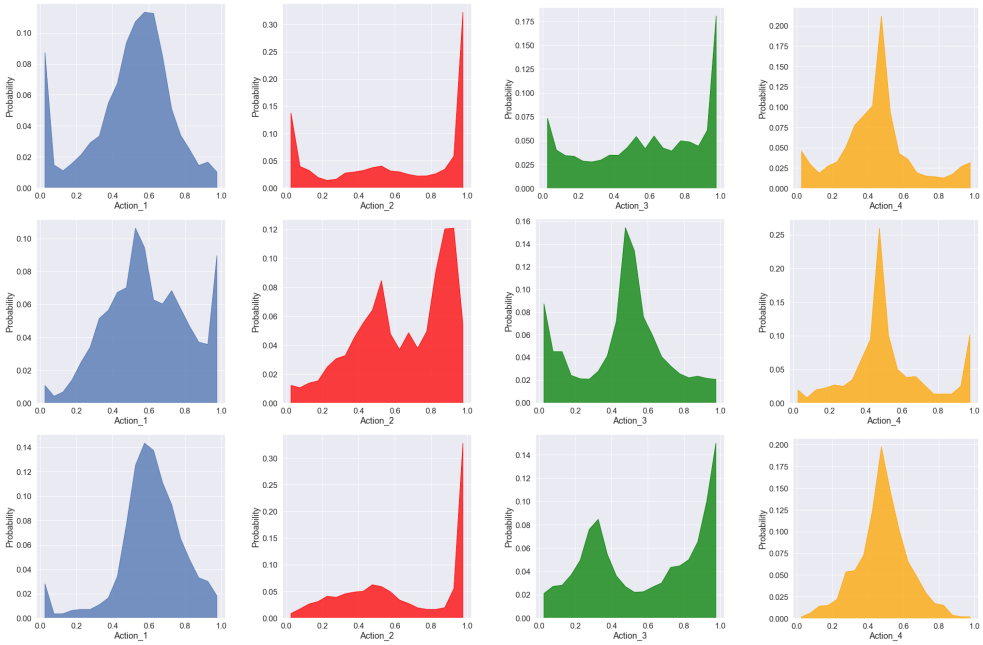


FIGURE 4.9: Action Distributions for (top row) SBT-ATEP, (middle row) FBT-ATEP, and (bottom row) EPOET40x40. Each column represents one specific dimension of the action array.

Chapter 5

Conclusion And Future Directions

Our research investigated the effect of having an augmentative topology agent on an Open-ended Learning algorithm's performance. Firstly, we asked the question that can augmentative topology agents learn endlessly through the framework of Enhanced POET (EPOET)? We showed that this is indeed the case by introducing Augmentative Topology Enhanced POET (ATEP) and by comparing it with EPOET, we found that our method outperforms EPOET on open-endedness via All New and Novel Environments Created and Solved (ANNECS) score. Secondly, we asked the question can we create a different transfer mechanism that aids more open-endedness and produces more generalized agents? We answer this question by experimenting with two different transfer mechanisms based on ATEP: Fitness-Based Transfer ATEP (FBT-ATEP) and Species-Based Transfer ATEP (SBT-ATEP). FBT-ATEP, which is inspired by EPOETs' transfer mechanism, slightly outperforms EPOET40x40 on open-endedness, which answers our first question, and performs as well as EPOET in generalization tests. Our second question is answered by SBT-ATEP, which is a novel transfer mechanism, inspired by the NeuroEvolution of Augmenting Topologies (NEAT) speciation mechanism, outperforms FBT-ATEP and EPOET on open-endedness and generalization tests while using fewer nodes in the neural networks, thus answering both of our questions and proving our hypothesis.

Our approach, however, does require more function evaluations than competing approaches. Thus, a promising future direction would be to use NEAT with Novelty Search [56] or Surprise Search which tends to converge faster than simple NEAT [38]. Quality-Diversity algorithms may also be worthwhile to explore in the context of Open-ended Learning as they have the ability to produce a population of high-performing and diverse individuals [69]. Exploring Neurogenesis [23, 61],

where neurons are added to a single neural network based on various external triggers, could also be a promising direction. To reduce computational load, it would also be promising to look into developing single-population Open-ended Learning methods without losing the exploration abilities of EPOET.

Furthermore, we have opened up possible future research into transfer mechanisms. We compared simple approaches such as FBT and SBT, but more advanced approaches could yield further performance improvements. For instance, we could combine both FBT and SBT in a weighted manner, or transfer only a certain percentage of a species or population. Finally, this work provides a starting point, like EPOET itself, into Open-ended Learning with augmentative topology agents. We, therefore, used the simple 2D BipedalWalker as our benchmark. Future work should compare ATEP with standard EPOET in different and more complex environments. Ultimately, we hope that this new approach furthers research into open-ended algorithms that do not slow down over time and can keep up with an ever-changing environment.

Bibliography

[1] Abbas Abdolmaleki et al. "Relative entropy regularized policy iteration". In: *arXiv preprint arXiv:1812.02256* (2018).

[2] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[3] Thomas Bäck and Hans-Paul Schwefel. "An overview of evolutionary algorithms for parameter optimization". In: *Evolutionary computation* 1.1 (1993), pp. 1–23.

[4] Matthew Barthet, Antonios Liapis, and Georgios N Yannakakis. "Open-Ended Evolution for Minecraft Building Generation". In: *IEEE Transactions on Games* (2022).

[5] Varun Bhatt et al. "Deep surrogate assisted generation of environments". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 37762–37777.

[6] Eric Bonabeau et al. *Swarm intelligence: from natural to artificial systems*. 1. Oxford university press, 1999.

[7] Jonathan C Brant and Kenneth O Stanley. "Minimal criterion coevolution: a new approach to open-ended search". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 67–74.

[8] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[9] Victor Campos et al. "Explore, Discover and Learn: Unsupervised Discovery of State-Covering Skills". In: *Proceedings of the 37th International Conference on Machine Learning*. Vol. 119. 2020, pp. 1317–1327.

[10] Rich Caruana and Alexandru Niculescu-Mizil. "An empirical comparison of supervised learning algorithms". In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 161–168.

[11] M Emre Celebi and Kemal Aydin. *Unsupervised learning algorithms*. Vol. 9. Springer, 2016.

[12] Xi Chen et al. "Variational lossy autoencoder". In: *arXiv preprint arXiv:1611.02731* (2016).

[13] Maria Cipollone, Catherine C Schifter, and Rick A Moffat. "Minecraft as a creative tool: A case study". In: *International Journal of Game-Based Learning (IJGBL)* 4.2 (2014), pp. 1–14.

[14] Maximilien Le Clei and Pierre Bellec. "Neuroevolution of recurrent architectures on control tasks". In: *Genetic and Evolutionary Computation Conference, Companion Volume*. Ed. by Jonathan E. Fieldsend and Markus Wagner. 2022, pp. 651–654.

[15] Cédric Colas et al. "Scaling map-elites to deep neuroevolution". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 2020, pp. 67–75.

[16] Antoine Cully et al. "Robots that can adapt like animals". In: *Nature* 521.7553 (2015), pp. 503–507.

[17] Zihang Dai et al. "Transformer-xl: Attentive language models beyond a fixed-length context". In: *arXiv preprint arXiv:1901.02860* (2019).

[18] Michael Dennis et al. "Emergent complexity and zero-shot transfer via unsupervised environment design". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 13049–13061.

[19] Aaron Dharna, Julian Togelius, and Lisa B Soros. "Co-generation of game levels and game-playing agents". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 16. 1. 2020, pp. 203–209.

[20] Tansel Dokeroglu et al. "A survey on new generation metaheuristic algorithms". In: *Computers & Industrial Engineering* 137 (2019), p. 106040.

[21] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. "Ant colony optimization". In: *IEEE computational intelligence magazine* 1.4 (2006), pp. 28–39.

[22] Kees Dorst and Nigel Cross. "Creativity in the design process: co-evolution of problem–solution". In: *Design studies* 22.5 (2001), pp. 425–437.

[23] Timothy J Draelos et al. "Neurogenesis deep learning: Extending deep networks to accommodate new classes". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 526–533.

[24] David B D'Ambrosio, Jason Gauci, and Kenneth O Stanley. "HyperNEAT: The first five years". In: *Growing Adaptive Machines: Combining Development and Learning in Artificial Neural Networks* (2014), pp. 159–185.

[25] Sam Earle et al. "Illuminating diverse neural cellular automata for level generation". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2022, pp. 68–76.

[26] Adrien Ecoffet et al. "First return, then explore". In: *Nature* 590.7847 (2021), pp. 580–586.

[27] A W F Edwards. "The Genetical Theory of Natural Selection". In: *Genetics* 154.4 (Apr. 2000), pp. 1419–1426. ISSN: 1943-2631. DOI: 10.1093/genetics/154.4.1419. eprint: https://academic.oup.com/genetics/article-pdf/154/4/1419/42025776/genetics1419.pdf. URL: https://doi.org/10.1093/genetics/154.4.1419.

[28] Benjamin Eysenbach et al. "Diversity is All You Need: Learning Skills without a Reward Function". In: *International Conference on Learning Representations*. 2019.

[29] Linxi Fan et al. "MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge". In: *arXiv preprint arXiv:2206.08853* (2022).

[30] Alhussein Fawzi et al. "Discovering faster matrix multiplication algorithms with reinforcement learning". In: *Nature* 610.7930 (2022), pp. 47–53.

[31] Sevan G Ficici and Jordan B Pollack. "Challenges in Coevolutionary Learning: Arms-Race Dynamics". In: *Artificial Life VI: Proceedings of the sixth international conference on artificial life*. Vol. 6. MIT Press. 1998, p. 238.

[32] Vincent François-Lavet et al. "An introduction to deep reinforcement learning". In: *Foundations and Trends® in Machine Learning* 11.3-4 (2018), pp. 219–354.

[33] Edgar Galván and Peter Mooney. "Neuroevolution in deep neural networks: Current trends and future challenges". In: *IEEE Transactions on Artificial Intelligence* 2.6 (2021), pp. 476–493.

[34] Chuang Gan et al. “OPEn: An Open-ended Physics Environment for Learning Without a Task”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 5878–5885.

[35] Fred Glover and Manuel Laguna. “Tabu search”. In: *Handbook of combinatorial optimization*. Springer, 1998, pp. 2093–2229.

[36] Faustino J Gomez, Risto Miikkulainen, et al. “Solving non-Markovian control tasks with neuroevolution”. In: *IJCAI*. Vol. 99. 1999, pp. 1356–1361.

[37] Scott R Granter, Andrew H Beck, and David J Papke Jr. “AlphaGo, deep learning, and the future of the human microscopist”. In: *Archives of pathology & laboratory medicine* 141.5 (2017), pp. 619–621.

[38] Daniele Gravina, Antonios Liapis, and Georgios Yannakakis. “Surprise search: Beyond objectives and novelty”. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. 2016, pp. 677–684.

[39] Frederic Gruau, Darrell Whitley, and Larry Pyeatt. “A comparison between cellular encoding and direct encoding for genetic neural networks”. In: *Proceedings of the 1st annual conference on genetic programming*. 1996, pp. 81–89.

[40] Danijar Hafner et al. “Mastering Diverse Domains through World Models”. In: *arXiv preprint arXiv:2301.04104* (2023).

[41] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[42] John H Holland. “Genetic algorithms”. In: *Scientific american* 267.1 (1992), pp. 66–73.

[43] John J Hopfield. “Artificial neural networks”. In: *IEEE Circuits and Devices Magazine* 4.5 (1988), pp. 3–10.

[44] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. “Prioritized level replay”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 4940–4950.

[45] Peiyuan Jiang et al. “A Review of Yolo algorithm developments”. In: *Procedia Computer Science* 199 (2022), pp. 1066–1073.

[46] John Jumper et al. “AlphaFold 2”. In: *In Fourteenth Critical Assessment of Techniques for Protein Structure Prediction (Abstract Book* (2020).

[47] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. "A review on genetic algorithm: past, present, and future". In: *Multimedia Tools and Applications* 80.5 (2021), pp. 8091–8126.

[48] James Kennedy and Russell Eberhart. "Particle swarm optimization". In: *Proceedings of ICNN'95-international conference on neural networks*. Vol. 4. IEEE. 1995, pp. 1942–1948.

[49] Miklos Kepes, Nicholas Guttenberg, and LB Soros. "ChemGrid: An Open-Ended Benchmark Domain for an Open-Ended Learner". In: *2022 IEEE Conference on Games (CoG)*. IEEE. 2022, pp. 221–228.

[50] Ahmed Khalifa et al. "Talakat: Bullet hell generation through constrained map-elites". In: *Proceedings of The Genetic and Evolutionary Computation Conference*. 2018, pp. 1047–1054.

[51] Khimya Khetarpal et al. "Towards Continual Reinforcement Learning: A Review and Perspectives". In: *CoRR* abs/2012.13490 (2020). arXiv: 2012.13490. URL: https://arxiv.org/abs/2012.13490.

[52] B Ravi Kiran et al. "Deep reinforcement learning for autonomous driving: A survey". In: *IEEE Transactions on Intelligent Transportation Systems* 23.6 (2021), pp. 4909–4926.

[53] Vijay Kumar, Jitender Kumar Chhabra, and Dinesh Kumar. "Parameter adaptive harmony search algorithm for unimodal and multimodal optimization problems". In: *Journal of Computational Science* 5.2 (2014), pp. 144–155.

[54] Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. "Grammar variational autoencoder". In: *International Conference on Machine Learning*. 2017, pp. 1945–1954.

[55] H Chad Lane and Sherry Yi. "Playing with virtual blocks: Minecraft as a learning environment for practice and research". In: *Cognitive development in digital contexts*. Elsevier, 2017, pp. 145–166.

[56] Joel Lehman, Kenneth O Stanley, et al. "Exploiting open-endedness to solve problems through the search for novelty." In: *ALIFE*. Citeseer. 2008, pp. 329–336.

[57] Timothée Lesort et al. “Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges”. In: *Inf. Fusion* 58 (2020), pp. 52–68. DOI: 10.1016/j.inffus.2019.12.004. URL: https://doi.org/10.1016/j.inffus.2019.12.004.

[58] Shutao Li et al. “Deep learning for hyperspectral image classification: An overview”. In: *IEEE Transactions on Geoscience and Remote Sensing* 57.9 (2019), pp. 6690–6709.

[59] Zhizhong Li and Derek Hoiem. “Learning Without Forgetting”. In: Lecture Notes in Computer Science 9908 (2016). Ed. by Bastian Leibe et al., pp. 614–629. DOI: 10.1007/978-3-319-46493-0_37. URL: https://doi.org/10.1007/978-3-319-46493-0_37.

[60] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. “Neuroevolutionary constrained optimization for content creation”. In: *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*. IEEE. 2011, pp. 71–78.

[61] Kaitlin Maile et al. “When, where, and how to add new neurons to ANNs”. In: *International Conference on Automated Machine Learning*. PMLR. 2022, pp. 18–1.

[62] Robert Meier and Asier Mujika. “Open-Ended Reinforcement Learning with Neural Reward Functions”. In: *arXiv preprint arXiv:2202.08266* (2022).

[63] Zbigniew Michalewicz and Marc Schoenauer. “Evolutionary algorithms for constrained parameter optimization problems”. In: *Evolutionary computation* 4.1 (1996), pp. 1–32.

[64] Tom M Mitchell. *Machine learning*. Vol. 1. 9. McGraw-hill New York, 1997.

[65] David J Montana, Lawrence Davis, et al. “Training feedforward neural networks using genetic algorithms.” In: *IJCAI*. Vol. 89. 1989, pp. 762–767.

[66] David E Moriarty and Risto Miikkulainen. “Forming neural networks through efficient and adaptive coevolution”. In: *Evolutionary computation* 5.4 (1997), pp. 373–399.

[67] Hirotaka Moriguchi and Shinichi Honiden. “CMA-TWEANN: efficient optimization of neural networks via self-adaptation and seamless augmentation”. In: *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. 2012, pp. 903–910.

[68] Jean-Baptiste Mouret and Jeff Clune. "Illuminating search spaces by mapping elites". In: *arXiv preprint arXiv:1504.04909* (2015).

[69] Olle Nilsson and Antoine Cully. "Policy gradient assisted map-elites". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2021, pp. 866–875.

[70] Emma Stensby Norstein, Kai Olav Ellefsen, and Kyrre Glette. "Open-Ended Search for Environments and Adapted Agents Using MAP-Elites". In: *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer. 2022, pp. 651–666.

[71] Charles Ofria and Claus O Wilke. "Avida: A software platform for research in computational evolutionary biology". In: *Artificial life* 10.2 (2004), pp. 191–229.

[72] Eirik Solheim Ølberg. "CHILD: Predicting simulation outcome in open-ended co-evolution". MA thesis. 2022.

[73] Keiron O'Shea and Ryan Nash. "An introduction to convolutional neural networks". In: *arXiv preprint arXiv:1511.08458* (2015).

[74] Johannes von Oswald et al. "Continual learning with hypernetworks". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=SJgwNerKvB.

[75] Norman Packard et al. "An overview of open-ended evolution: Editorial introduction to the open-ended evolution ii special issue". In: *Artificial life* 25.2 (2019), pp. 93–103.

[76] Jan Paredis. "Coevolutionary computation". In: *Artificial life* 2.4 (1995), pp. 355–375.

[77] German Ignacio Parisi et al. "Continual lifelong learning with neural networks: A review". In: *Neural Networks* 113 (2019), pp. 54–71. DOI: 10.1016/j.neunet.2019.01.012. URL: https://doi.org/10.1016/j.neunet.2019.01.012.

[78] Jack Parker-Holder et al. "Evolving curricula with regret-based environment design". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 17473–17498.

[79] Jigar Patel et al. “Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques”. In: *Expert systems with applications* 42.1 (2015), pp. 259–268.

[80] Martin L Puterman. “Markov decision processes”. In: *Handbooks in operations research and management science* 2 (1990), pp. 331–434.

[81] Alexander Quessy and Thomas Stuart Richardson. “Rewardless Open-Ended Learning (ROEL)”. In: *OpenReview* (2021).

[82] Nicholas J Radcliffe. “Genetic set recombination and its application to neural network topology optimisation”. In: *Neural Computing & Applications* 1.1 (1993), pp. 67–90.

[83] BR Rajakumar. “Static and adaptive mutation techniques for genetic algorithm: a systematic comparative analysis”. In: *International Journal of Computational Science and Engineering* 8.2 (2013), pp. 180–193.

[84] Thomas S Ray. “An approach to the synthesis of life”. In: *Artificial life II* (1991), pp. 371–408.

[85] Raul Rojas and Raúl Rojas. “The backpropagation algorithm”. In: *Neural networks: a systematic introduction* (1996), pp. 149–182.

[86] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016).

[87] Andrei A. Rusu et al. “Progressive Neural Networks”. In: *CoRR* abs/1606.04671 (2016). arXiv: 1606.04671. URL: http://arxiv.org/abs/1606.04671.

[88] Fereshteh Sadeghi and Sergey Levine. “Cad2rl: Real single-image flight without a single real image”. In: *arXiv preprint arXiv:1611.04201* (2016).

[89] Tim Salimans et al. “Evolution strategies as a scalable alternative to reinforcement learning”. In: *arXiv preprint arXiv:1703.03864* (2017).

[90] Jacob Schrum, Vanessa Volz, and Sebastian Risi. “CPPN2GAN: Combining Compositional Pattern Producing Networks and GANs for Large-Scale Pattern Generation”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2020, 139–147.

[91] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[92] Archit Sharma et al. “Dynamics-Aware Unsupervised Discovery of Skills”. In: *International Conference on Learning Representations*. 2020.

[93] Gene Sher. “DXNN: evolving complex organisms in complex environments using a novel tweann system”. In: *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*. 2011, pp. 149–150.

[94] L Soros and Kenneth Stanley. “Identifying necessary conditions for open-ended evolution through the artificial life world of chromaria”. In: *ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems*. MIT Press. 2014, pp. 793–800.

[95] Lee Spector, Jon Klein, and Mark Feinstein. “Division blocks and the open-ended evolution of development, form, and behavior”. In: *Proceedings of the 9th annual conference on genetic and evolutionary computation*. 2007, pp. 316–323.

[96] Felix Stahlberg. “Neural machine translation: A review”. In: *Journal of Artificial Intelligence Research* 69 (2020), pp. 343–418.

[97] Kenneth O Stanley. “Compositional pattern producing networks: A novel abstraction of development”. In: *Genetic programming and evolvable machines* 8.2 (2007), pp. 131–162.

[98] Kenneth O Stanley. “Evolving neural networks”. In: *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*. 2012, pp. 805–826.

[99] Kenneth O Stanley. “Why open-endedness matters”. In: *Artificial life* 25.3 (2019), pp. 232–235.

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

[101] Kenneth O Stanley, Joel Lehman, and Lisa Soros. “Open-endedness: The last grand challenge you’ve never heard of”. In: *While open-endedness could be a force for discovering intelligence, it could also be a component of AI itself* (2017).

[102] Kenneth O Stanley and Risto Miikkulainen. “Evolving neural networks through augmenting topologies”. In: *Evolutionary computation* 10.2 (2002), pp. 99–127.

[103] Vladimir Stanovov, Shakhnaz Akhmedova, and Eugene Semenkin. "Neuroevolution for parameter adaptation in differential evolution". In: *Algorithms* 15.4 (2022), p. 122.

[104] Emma Hjellbrekke Stensby, Kai Olav Ellefsen, and Kyrre Glette. "Co-optimising Robot Morphology and Controller in a Simulated Open-Ended Environment". In: *International Conference on the Applications of Evolutionary Computation*. 2021, pp. 34–49.

[105] Felipe Petroski Such et al. "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning". In: *arXiv preprint arXiv:1712.06567* (2017).

[106] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[107] Richard S Sutton and Andrew G Barto. "Reinforcement learning: an introduction MIT Press". In: *Cambridge, MA* 22447 (1998).

[108] Danesh Tarapore et al. "How do different encodings influence the performance of the MAP-Elites algorithm?" In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. 2016, pp. 173–180.

[109] Adaptive Agent Team et al. "Human-Timescale Adaptation in an Open-Ended Task Space". In: *arXiv preprint arXiv:2301.07608* (2023).

[110] Open Ended Learning Team et al. "Open-ended learning leads to generally capable agents". In: *arXiv preprint arXiv:2107.12808* (2021).

[111] Qazi Zia Ullah et al. "A cartesian genetic programming based parallel neuroevolutionary model for cloud server's CPU usage prediction". In: *Electronics* 10.1 (2021), p. 67.

[112] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[113] Christos Voudouris and Edward Tsang. "Guided local search and its application to the traveling salesman problem". In: *European journal of operational research* 113.2 (1999), pp. 469–499.

[114] Rui Wang et al. "Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 9940–9951.

[115] Rui Wang et al. “Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions”. In: *arXiv preprint arXiv:1901.01753* (2019).

[116] Darrell Whitley et al. “Genetic reinforcement learning for neurocontrol problems”. In: *Machine Learning* 13.2 (1993), pp. 259–284.

[117] Rudolf Paul Wiegand. *An analysis of cooperative coevolutionary algorithms*. George Mason University, 2004.

[118] Patrick Henry Winston. *Artificial intelligence*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[119] Larry S Yaeger and Olaf Sporns. “Evolution of neural structure and complexity in a computational ecology”. In: *Artificial Life X: Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems*. Citeseer. 2006, pp. 330–336.

[120] Michalewicz Zbigniew. “Genetic algorithms+ data structures= evolution programs”. In: *Computational Statistics*. Springer-Verlag, 1996, pp. 372–373.

[121] Fangqin Zhou and Joaquin Vanschoren. “Open-Ended Learning Strategies for Learning Complex Locomotion Skills”. In: *arXiv preprint arXiv:2206.06796* (2022).

Appendix A

Hyperparameter settings

TABLE A.1: EPOET general hyperparameter settings for ATEP

Hyperparameter	Setting
Reward threshold	200
Environment difficulty MC	25 - 340
Transfer check	25
Reproducibility check	150
Active environments	20

TABLE A.2: ES hyperparameter settings

Hyperparameter	Setting
ES Population	512
Weight update method	Adam
Initial learning rate	0.01
Decay factor of learning rate	0.9999
Initial noise standard deviation	0.1
Lower bound of noise standard deviation	0.01
Decay factor of noise standard deviation	0.999

TABLE A.3: NEAT hyperparameter settings

Hyperparameter	**Setting**
Population size	1000
Crossover probability	0.3
Weight mutation (small) probability	0.85
Weight mutation (large) probability	0.15
Weight mutation (small) range	-0.1 - +0.1
Weight mutation (large) range	-1 - +1
Connection mutation probability	0.85
Node mutation probablity	0.15
Maximum stagnation	60
c_{1}	1.0
c_{2}	1.0
c_{3}	3.7
Delta threshold	3.0
Initial condition	full
Activation function	tanh
Number of inputs	24
Number of outputs	4

TABLE A.4: CPPN hyperparameter settings

Hyperparamer	**Setting**
Initial condition	full
Activation default	identity
Activation options	identity sin sigmoid square tanh
Aggregation default	sum
bias init stdev	0.1
bias init type	gaussian
bias max value	10.0
bias min value	-10.0
bias mutate power	0.1
bias mutate rate	0.75
num inputs	1
num outputs	1
response init mean	1.0
response init type	gaussian
response max value	10.0
response min value	-10.0
single structural mutation	True
structural mutation surer	default
weight init stdev	0.25
weight init type	gaussian
weight max value	10.0
weight min value	-10.0
weight mutate power	0.1
weight mutate rate	0.75