Jakob Myklebust Vaardal

Exploring the Paired Open-Ended Trailblazer algorithm

Master's thesis in Computer Science

July 2022

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Jakob Myklebust Vaardal

Exploring the Paired Open-Ended Trailblazer algorithm

Master's thesis in Computer Science
July 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

Life on earth displays an explosion of creativity and diversity. One single run of evolution has managed to create both photosynthesis, flight and human intelligence; and is still presenting us with new solutions to the problem of survival and reproduction on earth. This capability for never-ending creation of novel organisms is what the field of open-ended evolution is trying to replicate.

In this work, we will explore the Paired Open-Ended Trailblazer (POET) algorithm, which belongs to the field of open-ended evolution. POET is a coevolutionary algorithm seeking to endlessly generate problems of increasing difficulty and their increasingly complex solutions through the enforcement of a minimal criterion and goal-switching. Previously POET has been shown to generate and solve complex problems in robot locomotion control and general game-playing environments [21] [29]. Challenges that were solved in the robot locomotion environment were found not solvable by direct optimization or direct-path curriculum building.

We apply POET to a new type of reinforcement learning environment and combine POET with NeuroEvolution of Augmenting Topologies (NEAT) for the first time. The novel environment serves as an exciting new playground for exploring POET's ability to create and solve problems of increasing complexity. At the same time, NEAT allows artificial neural network controller topology to increase in complexity incrementally.

Our model is shown to generate a diverse curriculum of increasing complexity while solving many of the generated challenges. During the model's coevolutionary runs, we observe open-ended tendencies, if there are any. We also importantly perform an ablation study to investigate essential POET components like the minimal-criterion and goal-switching. Excitingly, we find that some parts of the POET algorithm seem not to be as important as previously believed.

Sammendrag

Livet på jorden vitner til en eksplosjon av kreativitet som har skapt et enormt mangfold av ulike organismer. Ved hjelp av ett løp med evolusjon har egenskaper som både fotosyntese, flyvning og menneskelig intelligens oppstått. Selv om evolusjon har fylt verden med utallige komplekse skapelser, stopper den ikke der. Evolusjon fortsetter å presentere oss med nye svar til spørsmålet om overlevelse og reproduksjon for hver eneste generasjon. Denne egenskapen for tilsynelatende uendelig skapning av nye utfordringer og deres løsninger er hva feltet åpen evolusjon forsøker å etterligne.

Masteroppgaven utforsker algoritmen Paired Open-Ended Trailblazer (POET) som tilhører feltet åpen evolusjon. POET er en ko-evolusjonær algoritme som søker å uendelig generere problemer med økende vanskelighetsgrad og deres stadig mer komplekse løsninger. Vi anvender POET i et nytt forsterkende læringsmiljø og kombinerer POET med NeuroEvolution of Augmenting Topologies (NEAT) for første gang. Det nye miljøet fungerer som en spennende ny lekeplass for å utforske POETs evne til å skape og løse problemer med økende vanskelighetsgrad, mens NEAT lar topologien til de kunstige nevrale nettverks løsningene til å gradvis kompleksifiseres.

Ved å bruke vår modell blir en mangfoldig læreplan med økende kompleksitet generert, samtidig som den løser mange av de genererte utfordringene. Under de ko-evolusjonære kjøringene observerer vi konvergens og åpne egenskaper om det er noen. En ablasjons studie er også utført for å undersøke viktige POET-komponenter som minimal-kriteriet og målbytte. Et spennende funn er at noen komponenter i POET-algoritmen ikke ser ut til å være så viktige som tidligere antatt.

Preface

The work herein constitutes a master's thesis in Computer Science at the Norwegian University of Science and Technology. The literature search was conducted during the Specialization Project in the spring of 2021, where most of the knowledge was accumulated. Implementation of our model and experiments were conducted during the spring of 2022. The master's thesis explores the Paired Open-Ended Trailblazer algorithm while using NeuroEvolution of Augmenting Topologies for agent optimization in the reinforcement learning environment Super Mario Bros.

I thank my supervisor Professor Pauline Haddow for excellent guidance throughout the process.

Jakob Myklebust Vaardal
Tollaksøya, July 25, 2022

Abbreviations

- ANN - Artificial Neural Network
- EE - Environmental Encoding
- EC - Environmental Characterization
- RL - Reinforcement Learning
- MDP - Markov decision process
- POMDP - Partially observable Markov decision process
- EA - Evolutionary Algorithm
- GA - Genetic Algorithm
- ES - Evolutionary Strategies
- NE - Neuroevolution
- NEAT - NeuroEvolution of Augmenting Topologies
- HyperNEAT - Hypercube-based NEAT
- ES-HyperNEAT - Evolvable Substrate HyperNEAT
- MCC - Minimal Criterion Co-Evolution
- POET - Paired Open-Ended Trailblazer

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Finding a genuinely open-ended algorithm has not yet been achieved. Several attempts have been made, but none of them are able to emulate nature's ability of seemingly never-ending creativity. Most work in artificial intelligence today focuses on minimizing a loss function through machine learning techniques such as gradient descent and backpropagation. These methods have given us impressive tools for challenges like image classification, natural language processing, and robotics. However, traditional machine learning approaches converge towards a single solution and are not capable of significant further learning and improvement. Instead, approaches focused on open-endedness would like to see a continual discovery of new problems of increasing complexity and their solutions. Mimicking how evolution provides a never-ending stream of novel answers to the question of survival and reproduction.

Data has been the key to creating increasingly complex machine learning models throughout machine learning history. Tech companies and researchers have an unquenchable thirst, as each new data point may improve the performance of their models. Therefore, a natural question was whether the algorithms themselves could generate the data needed. During the last decade, the machine learning community has achieved remarkable breakthroughs by using methods that can do just that. Self-play is one such approach, generating data by making the model play against itself. For example, through solely using self-play AlphaZero was able to beat the previous chess world champion Stockfish [21]. Generative adversarial networks produced similarly impressive results for image classification by making two neural networks compete in a zero-sum game [9]. These two approaches draw inspiration from evolutionary biology, igniting an arms race between artificial neural networks.

The Paired Open-Ended Trailblazer (POET) algorithm was created to gener-

ate and solve problems of increasing complexity in an attempt to achieve open-endedness [29]. Previously the POET algorithm has been successfully used to generate curricula for robot locomotion automatically, and general game-playing [29] [25]. It is a novel coevolutionary algorithm that employs two interlocked populations, a population of environments and a population of agents. By evolving the two populations in parallel, it generates its curriculum automatically. Through the use of a minimal-criterion and optimization of agents, the two populations gradually shift toward greater complexity.

1.1 Goals and Research Questions

Goal *Explore the Paired Open-Ended Trailblazer algorithm*

Research question 1 *How important is the minimal-criterion in the Paired Open-Ended Trailblazer algorithm?*

Research question 2 *How important is the transfer component of the Paired Open-Ended Trailblazer algorithm?*

Research question 3 *Is the Paired Open-Ended Trailblazer algorithm combined with Neuroevolution of Augmenting Topologies for agent optimization able to solve Super Mario Bros levels of high difficulty?*

Research question 4 *Does the Paired Open-Ended Trailblazer algorithm facilitate learning better than direct optimization for reinforcement learning?*

1.2 Research Method

The research methodology of this project is an analytical process. The reasoning will be based on the knowledge and data found by using the structured literature review protocol.

1.3 Structured Literature Review Protocol

This section presents the structured literature review protocol used to find relevant literature. Section 1.3.1 describes the search process for identifying literature to be considered. Section 1.3.2 describes the selection criteria used to decide if a research article should be included.

1.3.1 Search process

Two approaches were applied to discover relevant material. The primary strategy was to use the search engine *Google Scholar*. When searching for material, the keywords *Open Ended Evolution, Coevolution, Minimal Criterion, Neuroevolution* and *Paired Open Ended Trailblazer* were used. The second approach was to discover material by exploring research articles cited by material found through *Google Scholar*.

1.3.2 Selection Criteria

Literature is deemed relevant if it satisfies the inclusion criteria (IC) and the quality criteria (QC). The following criteria were used to select research articles from the search process.

IC *The research article focuses on a relevant application or method within open-ended evolution*

QC *Statements and claims in the article should be supported by results or a source*

QC *The authors are critical to their own results*

1.4 Contributions

The master's thesis has four main contributions. The most significant contribution is the study of the minimal-criterion and goal-switching in the Paired Open-Ended Trailblazer (POET) algorithm. An ablation study was performed to determine whether the algorithm's minimal criterion and transfer mechanisms are necessary. During the ablation study, different thresholds for the minimal-criterion and transfer frequencies were also explored to determine how they affect the coevolutionary process.

The work also contributes by applying POET to a new problem domain. There have been limited previous applications of POET. In this work, we use POET to successfully solve challenging control problems with maze-like characteristics while at the same time introducing adversaries. Problems solved and generated by POET for the new problem domain were found to pose a significant challenge even for advanced human players. We also contribute by using the Neuroevolution of Augmenting Topologies (NEAT) for agent optimization in POET. To our knowledge, this is the first time POET has been combined with

an optimization method that searches for artificial neural network topology and connection weights at the same time.

Previous work drawing inspiration from POET by Uber AI labs and Google DeepMind has utilized vast computing resources [25] [29]. In this work, we explored using POET with limited resources by studying how resources should be distributed. We also explored whether POET is applicable when employing a limited amount of agent-environment pairs. The knowledge accumulated in this work may be used to guide future works where large-scale distributed processing is not available.

The final contribution is a POET framework implemented in Java. The framework will be available in a public git repository which can be found at https://github.com/jakobvaa.

Chapter 2

Background Theory

2.1 Machine Learning

Machine learning is a field of study focusing on allowing computers to learn by themselves. The field is seen as a part of artificial intelligence, which attempts to demonstrate intelligence through machines. Machine learning has been applied to many problems, like medicine, robotics, speech recognition and computer vision.

By learning from data, the machine learning algorithms can improve their performance in solving a set of tasks. The algorithms learn from training data to build a model which is used to make predictions or decisions without being explicitly programmed. The machine learning model is similar to a mathematical function. When given a value x, it outputs a value y, like the function y=f(x). The model gradually improves through experience from training data or interaction with simulated environments to output better predictions or decisions.

Machine learning approaches are often divided into three categories depending on the nature of the feedback available to the algorithm. Supervised learning concerns the problems where the algorithm can learn from input data and their desired outputs. The model attempts to approximate a function that maps input data to their correct output. Unsupervised learning concerns problems where only input data is available, the algorithm having to discover structures in the input without knowing its correct label. Reinforcement learning is the last category, which concerns problems where the program has to learn from interacting with an environment.

2.1.1 Reinforcement Learning

Reinforcement learning is machine learning that focuses on how software agents should act when interacting with an environment to maximize its cumulative reward. The field is also studied in many other disciplines, like control theory, simulation-based optimization, and game theory. In reinforcement learning, the agents are typically evaluated in environments represented by a Markov decision process. An example of reinforcement learning is autonomous driving, where an agent must learn to control a vehicle through trial and error.

2.2 Markov decision process

In mathematics, a Markov decision process (MDP) is a control process with discrete-time steps. It provides a mathematical framework for decision-making problems where the outcomes result from stochasticity and the decisions of the decision maker. They are used in many disciplines, like robotics and automatic control.

A Markov decision process consists of 4 different components, a set of states, a set of actions available from each state, a state transition function, and the reward function. In reinforcement learning, the state transition function or reward function is unknown. Through interaction with a simulator which can be formalized as an MDP, the reinforcement learning agent learns how to maximize its cumulative reward.

The system dynamics of some decision processes are assumed to be determined by an MDP, but the agent can not fully observe the underlying state. Such problems are called partially observable Markov decision processes (POMDP). When working with a POMDP, the goal is to find an action policy that returns the optimal action without certainly knowing the current state. Typical examples of POMDPs are robot navigation problems and planning under uncertainty.

2.3 Artificial Neural Networks

Artificial neural networks (ANN) are computing systems that are inspired by biological brains. An ANN consists of nodes that are linked through weighted connections. The nodes are the artificial equivalent of a neuron, while the connections correspond to synapses. Instead of transmitting an electrical signal to the next neuron, the nodes transmit a numerical value through the connections. A small neural network with a few nodes and connections is in itself not very expressive. However, by using one hidden layer, neural networks are universal function approximators by Cybenko's theorem [5].

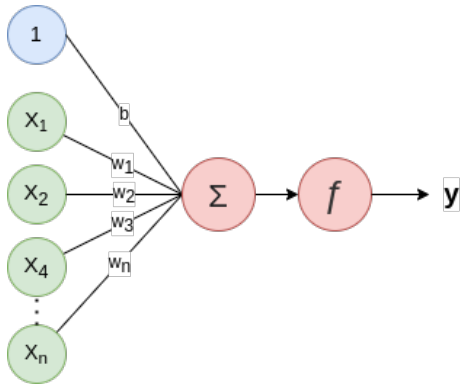

Figure 2.1: A perceptron. Input values are multiplied by their corresponding weight, then the sum of all weighted inputs is calculated along with the weighted bias. The sum is passed into the activation function, outputting a value y.

Figure 2.1 displays a single perceptron. Perceptrons are the building blocks of artificial neural networks. A perceptron is a single node with incoming connections which maps an input vector to an output value. Every input value is multiplied by its corresponding connection weight. Afterward, the sum of every weighted input value and the bias is calculated and passed into an activation function. This simple computing unit can do linear classification, meaning it can decide if a vector belongs to one of two classes.

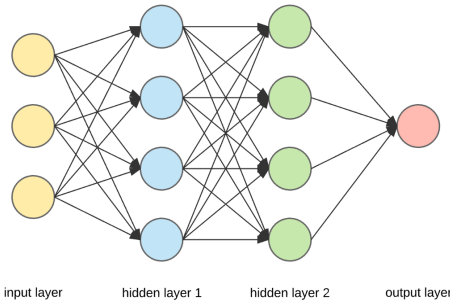

Figure 2.2: Example of a fully connected deep neural network. Deep neural networks with an input layer, two hidden layers, and an output layer. Typically deep neural networks have more than two hidden layers.

Perceptrons are typically arranged in layers that are connected. Figure 2.2 displays a fully connected neural network with four layers. Fully connected neural networks are a common choice of topology, where every node in each layer is connected to every node in the next layer of the network. The layers are separated into three parts, the input layer, hidden layers, and the output layer. Input data, for example, an image, is sent into the input layer, and the result of the input layer is passed on to the next layer. The result of each layer is calculated until the output layer produces an output. The output is typically some prediction, for example, whether an image contains an owl or not.

Training of neural networks is typically done through supervised learning, utilizing large amounts of training data.

2.4 Deep Learning

Deep learning is a part of a family of machine learning methods that are based on artificial neural networks. Deep learning architectures have been applied with major success in several different fields. Architectures such as deep neural networks, recurrent neural networks, and convolutional neural networks have been applied to computer vision, drug design, and board games with great success [14].

The adjective deep in deep learning refers to using multiple layers in the neural network. It was early shown that perceptrons are not capable of being a universal classifier, but if a network contains a non-polynomial activation function and a single hidden layer of unbounded width, it can represent any function.

Deep learning generally uses backpropagation and gradient descent to improve the artificial neural network. Through backpropagation, the gradient of the loss

function is calculated, while gradient descent is used to optimize ANN weights through incremental steps in the opposite direction of the gradient. The loss function in backpropagation is a function of the models parameters which determines how incorrect the model is in its prediction. Through backpropagation and gradient descent, the model is gradually improved by changing the weights incrementally towards a less incorrect model. For this to work, the loss function has to be differentiable.

There is no specified limit to how many hidden layers an ANN needs to have to be classified as a deep neural network, some would consider a network with one hidden layer as deep. However, the trend of deep learning is that the networks are getting deeper. When most refer to deep learning today, they usually refer to a network with more than a few hidden layers.

2.5 Genetic Algorithm

Genetic algorithms are search methods belonging to the class of evolutionary algorithms. Methods belonging to the field of evolutionary algorithms are inspired by Charles Darwin's theory of evolution, where individuals with favorable characteristics are able to survive and reproduce [6]. Genetic algorithms use the idea of natural selection to evolve a population of genomes. After successive generations of selecting genomes with desirable characteristics for reproduction, the population evolves towards optimal solutions. Genetic algorithms are typically used to find high-quality solutions for optimization and search problems.

In genetic algorithms, each individual of the population is represented by a set of genes, the genotype. The genome is mapped into a phenotype, which is how the genome is expressed in the environment. This is inspired by how DNA is indirectly mapped to the features of an organism.

Every generation candidate solutions of the population are selected for reproduction, based on how fit they are combined with stochasticity. Each individual's fitness is determined by how good of a solution the phenotype is to the problem being optimized for. When an individual is selected for reproduction, its genome is recombined by crossover and possibly mutation. A simple example of mutation in genetic algorithms can be to flip an arbitrary bit of the genetic sequence by some low probability.

2.6 Neuroevolution

The machine learning field has mainly focused on deep learning in recent years, where the training of neural networks is done through stochastic gradient descent and backpropagation. The field of neuroevolution is an alternative approach

using evolutionary algorithms to optimize artificial neural networks. By using algorithms inspired by evolution, the methods can discover connection weights, hyperparameters, and topology at the same time. Neuroevolution also has the added benefit of not utilizing a loss function, making it well suited for reinforcement learning where no differentiable loss function is available.

The simplest form of neuroevolution is conventional neuroevolution (CNE). The topology of the neural network is fixed, and only the weights of connections are evolved. Individuals are subject to mutation and crossover to generate offspring as in a simple genetic algorithm. A method called speciation is also often applied, which is used to maintain diversity in the population. CNE can fine tune each weight independently, resulting in high-performing policies if they exist within the policy space of the fixed topology neural network [4].

2.7 Neuroevolution of Augmenting Topologies

Biological brains do not have a static topology. Through evolution, it changes with every generation. Through neuroplasticity it changes during a lifetime of learning and forgetting. By augmenting topology, we may better imitate evolution's discovery of biological neural networks. The success of the convolutional neural network architecture is an example of how important topology is. By allowing topology to be discovered by the search process, structures like convolution might arise without using human intuition and craftsmanship.

Neuroevolution of augmenting topologies (NEAT) is a genetic algorithm that searches for artificial neural network topology along with connection weights [24]. As such, it represents an advance from methods that only optimize weights. Controlling topology allows NEAT to scale the complexity of the network to match that of the problem. Problems of low complexity are likely to require few hidden nodes, while more complex problems would require the network to increase in complexity. The NEAT algorithm is based upon three key components: tracking genes through historical markings to allow crossover of topology, speciation to protect the new structures, and incremental complexification of network topology from minimal initial structures.

NEAT is initialized by creating a population of minimal neural networks with zero hidden nodes. New nodes and connections are then introduced incrementally as structural mutations occur.

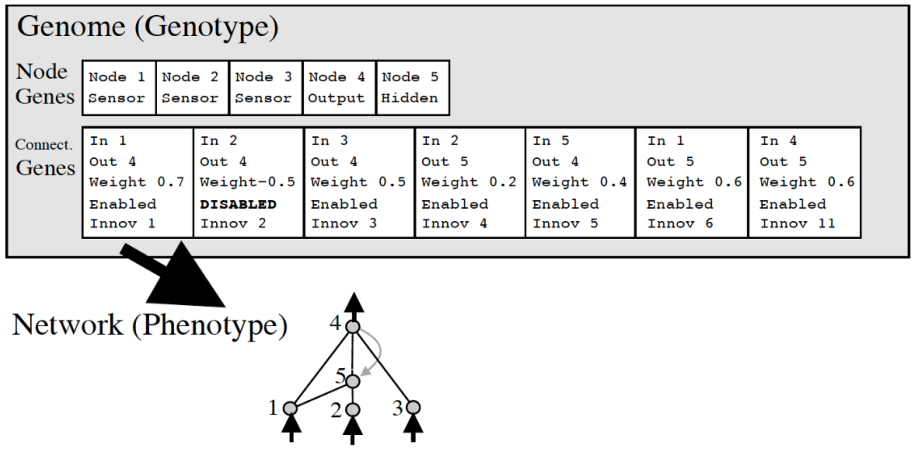

Figure 2.3: An example of a NEAT genotype to phenotype mapping. The artificial neural network has one input layer, one hidden node, and one output node. Seven connections are specified in the genotype, but only six are activated. This is because gene number two is disabled. The historical markings are stored as an innovation value in each connection gene. Figure from "Evolving Neural Networks through Augmenting Topologies" by Stanley 2002 [24].

Figure 2.3 displays NEAT's mapping from genotype to phenotype. Each neural network of the population is directly encoded by a genome consisting of connection genes. The connections genes refer to two node genes that are connected. Node genes encode input nodes, hidden nodes, and output nodes of the neural network. There are two different types of mutation, mutation of weights and topology. When a connection weight is mutated, it is perturbed by chance, as in other neuroevolution approaches. When the topology is mutated, either a node can be added to the genome or a connection. To minimize the mutation's initial effect, the weight of the new connection is 1.

As the individuals of the population are mutated, the neural network complexifies into topology of different sizes and structures. To be able to crossover the topology of the neural networks, historical markings are used. Whenever a new gene is added to a genome, a global innovation counter is incremented and stored along with the gene. This innovation number is called a historical marking. When the crossover of two networks is performed, the historical markings are used to line up genes. Two genes with the same historical marking must

represent the same structure as they are derived from the same ancestor gene. Figure 2.4 displays the crossover of two genomes.

A population of diverse neural networks of differing topology may form through mutation and crossover. However, topological innovations are often not retained, as they may reduce fitness. Smaller topologies optimize faster as the search space is reduced, and adding new nodes or connections may decrease fitness initially. Therefore speciation is applied to protect innovation. Speciation allows neural networks to compete within their own niches instead of competing with the entire population. By using speciation, topological innovations are protected and given time to be optimized. To determine which neural networks are in the same species, the historical markings are used to determine how similar the network topologies are.

Through mutation of connections, NEAT is able to evolve recurrent neural networks. Recurrent networks are not directed acyclical graphs like feedforward neural networks, which represent reactive controllers without state. On the other hand, a recurrent neural network has an internal state due to the recurrent connections. The state may serve as a memory that is useful for exploring partially observable Markov decision processes (POMDP). For example, in the challenging Atari 2600 game Montezuma's Revenge, it is useful for an agent to remember its previous actions. The recurrent connections have also previously allowed NEAT to solve non-Markovian problems like pole balancing [8].

NEAT has been applied to a variety of game domains, for example, controlling a team of robots in the NERO game and controlling a simulated car in The Open Car Racing Simulator [3]. The algorithm was also applied to general Atari game playing, which is a challenge where agents learn how to solve many different games. For general game playing, NEAT outperformed indirectly encoded neuroevolution methods like HyperNEAT for low-dimensional, noise, and object representations [10]. However, NEAT was found to struggle with high-dimensional input.

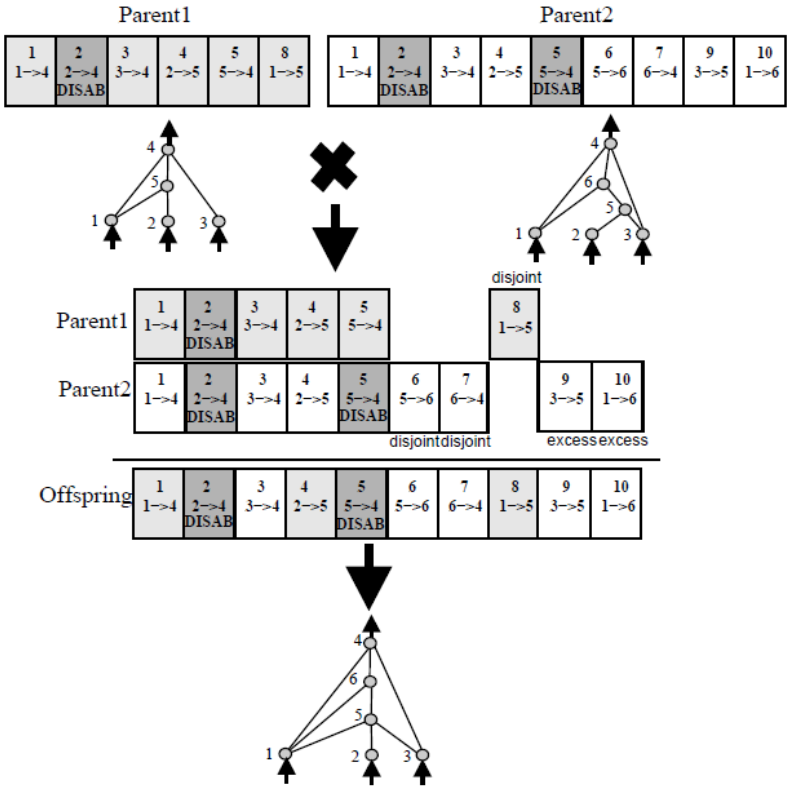

Figure 2.4: Example crossover of two genomes. Although Parent1 and Parent2 are different, historical markings can be used to tell us which genes line up with which. Matching genes are inherited randomly, while disjoint and excess genes are inherited from the more fit parent. In the example, equal fitness is assumed. Therefore disjoint and excess genes are inherited randomly. Figure from "Evolving Neural Networks through Augmenting Topologies" by Stanley 2002 [24].

2.8 Co-Evolution

Co-evolutionary algorithms typically refer to evolutionary algorithms where two or more populations are evolved simultaneously with coupled fitness [20]. The algorithms are divided into two categories, competitive and cooperative co-evolution. In competitive co-evolution, the fitness of an individual in one population is based on direct competition with individuals from another population. While in cooperative co-evolution, interacting individuals fail or succeed together.

2.9 Minimal Criterion Co-Evolution

Minimal-criterion coevolution (MCC) evolves two populations in parallel while enforcing a minimal-criterion on each individual [2]. Due to the minimal criterion, MCC does not fall into the traditional competitive or cooperative coevolution categories. This means that individuals are not rewarded based on the success or failure of others.

For an individual of one population to be eligible for reproduction, it has to satisfy a minimal-criterion with regard to the other population. An example of minimal-criterion co-evolution could be to evolve a population of maze problems in parallel with a population of maze solvers while enforcing a minimal-criterion on each population [2]. For a maze to be allowed to reproduce, it would have to be solved by at least one maze solver. Likewise, a maze solver would have to solve at least one maze to be eligible for reproduction.

There is no ranking among individuals which are able to satisfy the minimal-criterion. Therefore MCC stores the populations in a fixed-size queue, where individuals are ordered by time of insertion. To select the next individual for reproduction, an indexing number points to the individual next in line. After an individual has created an offspring, the child might be able to satisfy the minimal-criterion. If the minimal-criterion is satisfied, the offspring are added to the fixed-size population queue, pushing out the oldest individual if the queue is full. The intention of pushing the oldest individuals is to drift the population towards greater complexity. Through this setup, MCC ensures that every individual satisfying the minimal-criterion is allowed to reproduce at least once. The agnosticism of the selection method is intended to keep many divergent paths open simultaneously without convergence, as long as they satisfy the criterion [2].

2.10 Paired Open Ended Trailblazer

Minimal-criterion coevolution (MCC) was shown to continually generate novel, diverse, and increasingly complex maze problems along with their solutions in a single run which might hint at open-endedness [2]. However, there is no force for optimization within each environment in MCC. This means there is no pressure to improve a solution after it satisfies the minimal-criterion.

The paired open-ended trailblazer (POET) algorithm aims to replicate the open-ended nature of evolution by evolving a set of increasingly complex problems while optimizing their solutions in parallel. The algorithm draws inspiration from the minimal-criterion coevolution (MCC) algorithm by evolving a population of environments and a population of individuals in parallel while enforcing a minimal-criterion. In MCC, a problem is accepted into the population if it is solved by an individual of the other population; in POET, a problem is instead admitted if it seems likely to be a valuable stepping stone and solved after optimization. The intention is to create a swifter path toward solving increasingly complex challenges.

The fundamental algorithm of POET is simple. A population of environments and a population are evolved in parallel, the environments stored in a fixed-size queue. When the algorithm is initialized, a simple environment (e.g. a flat obstacle course) and a single agent (e.g. a random weight vector used as parameters for an artificial neural network) are added to their respective populations. These two initial individuals are paired, making up an agent-environment pair. During evolution, a list of environment-agent pairs is maintained at all times, consisting of several environments paired to each their own agent.

Each iteration of the POET main loop has three main tasks that are performed. The first task is to generate new environments. Every n generations, POET attempts to generate new environments from currently active environments and add them to the environment population. The second task is to optimize agents for their paired environments. Each paired agent is optimized for their respective environment every generation. The last task is to transfer agents. Every k generations agents are attempted transferred, if successful replacing the currently paired agent.

The generation of new environments is key to the curriculum-building properties of POET. To add a new environment, an individual environment queue is selected. The environment, which is an environmental encoding, is then mutated by random perturbations. The encoding of an environment could, for example, be a parameter vector describing the probability of different types of obstacles in an obstacle course. If the agent paired with the parent environment is able to satisfy a minimal-criterion for the new environment, it is added to the population and paired with a copy of the agent. The job of the minimal-criterion is to make

sure that the environments added are not too hard and not too easy.

By ensuring that the environments are of appropriate difficulty, the curriculum that emerges is intended to be smooth and well-calibrated for further learning. The minimal-criterion of POET is not as strict as the criterion of MCC. MCC demands every solution to solve at least one of the environments and every problem to be solved by at least one solution. In POET, the minimal-criterion is determined by agent progress towards solving an environment. An example minimal-criterion could be that agents need to at least achieve thirty percent of the maximum performance and no more than eighty percent of maximum performance. The idea is that if the agent is not able to achieve thirty percent, it indicates that the environment is too difficult to be explored at this time. If the agent achieves more than the upper limit of the minimal-criterion, it indicates that the level is too easy and will not enable further learning significantly.

When an agent is able to satisfy the minimal-criterion for a mutated environment, it is added to the environment population. Environments are stored in a fixed-size queue, putting a threshold on how many environments are optimized for simultaneously. When new environments are added to the queue, the oldest environments are removed to make room. The queue allows all environments to be explored equally as long as they have been admitted into the curriculum by satisfying the minimal-criterion. This means that environments do not disappear unless it is necessary, giving agents time to optimize for their paired environments and allowing skills learned to be transferred to other environments.

Every generation of the POET loop optimizes all agents. The goal is that every agent should improve towards solving their paired environment. Previously the optimization has been done through Evolution Strategies (ES), but any reinforcement learning algorithm can be applied [29]. The objective of the optimization is to maximize the performance achieved by the agents. Any performance measure can be used, for example, to maximize the distance traveled for an agent in an obstacle course environment or a simulated car for autonomous driving.

The last important part of the POET algorithm is the transfer step. Behaviors learned in one environment may be used as a stepping stone to progress toward solving another environment. Therefore agents are every k generations attempted transferred, replacing worse performing agents with the best agent for each environment. This may also allow the search process to escape from local optima, a common problem in machine learning. To decide which agents should be transferred, all agents are evaluated against all active environments. Each environment is then paired with its best-performing agent.

The POET algorithm is also easily parallelizable due to both optimization of agents and transfer of agents being independent operations. This allows POET to scale by utilizing the ongoing growth in available computing resources. Previous

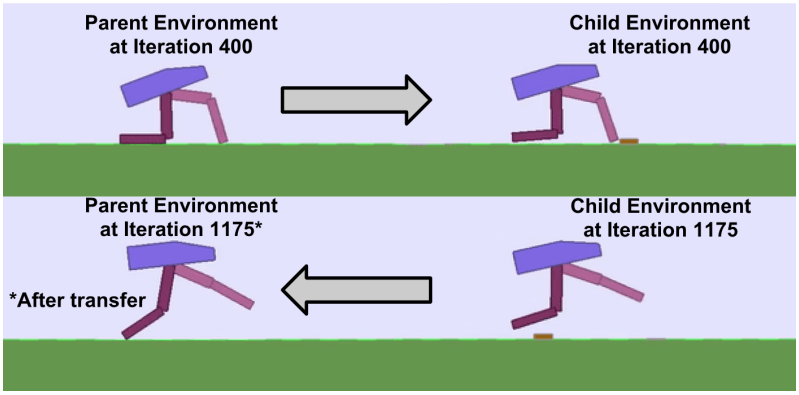

Figure 2.5: Agents evolved in the bipedal walker environment with the POET algorithm. The figure first shows how the agent is stuck in a local optimum at iteration 400, not having learned how to stand upright yet. The POET algorithm mutates the environment and optimizes the agent for the child environment where an obstacle is present. To overcome the obstacle, the agent learns a better walking gait. When the agent is transferred back to the parent environment, the behavior is improved. This shows us how environments may be used as stepping stones to achieve better performance on other problems than the one being optimized for. Figure from "Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions" by Wang et al. 2019

.

work has applied POET to the 2D bipedal-walker environment while harnessing the computing power of 256 parallel CPU cores [29].

2.11 Super Mario Bros

Super Mario Bros is a platform game developed by Nintendo and is one of the most influential video games of all time. In Super Mario Bros the player takes the role of Mario, guiding him through levels of varying difficulty.

Super Mario Bros gameplay involves moving the player-controlled character Mario through a two-dimensional obstacle course. The level is viewed from the side as shown in Figure 2.6. At each timestep, the player can activate five different moves making up a total of 32 (2^5) actions. The moves which can be activated are jump, sprint/fireball, left, right, and down.

Figure 2.6: Mario level containing three different types of obstacles and one type of enemy. To solve the level, the player has to move the Mario character from the left part of the level to the flagpole on the right-hand side. As the player can only see part of the level, the flagpole is not visible yet. The camera follows Mario, and as Mario moves to the right, more of the level is revealed.

To solve a Mario level, the player has to control Mario from the left side of the level to a goal flag on the right-hand side of the level. While moving through the level, Mario faces a rich variety of obstacles and adversaries that must be overcome.

The Mario game is a partially observable Markov decision process (POMDP). This means that the underlying process is an MDP, but the entire game state can not be observed. A player can observe a sliding window that is centered around the Mario character. As Mario moves to the right, the camera follows along, gradually revealing more of the level.

2.11.1 Levels

The game levels of the Super Mario Bros game can vary from simple to extreme complexity. Some levels even pose a great challenge to trained human players. The levels can be seen as a composition of many small challenges, which together can represent complex problems. There are three main challenges a Mario level consists of, gaps, adversaries, and obstacles.

The first challenge is gaps in the floor of each level. If Mario were to fall into a gap, the game would end. Jumping across a gap may be challenging, depending on the surrounding parts of the level. A level may consist of several gaps placed throughout the map. These gaps may have differing lengths, the longest gaps requiring the player to time when it jumps precisely. In Figure **??** the second level contains a gap which Mario can fall into. Some Mario levels consist of one large gap, with no floor to walk on. On these levels, the player has to navigate across blocks floating in the air. These levels pose a major challenge to expert players when combined with adversaries.

The second problem a player has to overcome is adversaries. Super Mario Bros has several types of adversaries which have different behaviors. There are five different types of adversaries, each with its own game mechanics. The five types are Goombas, green turtles, red turtles, spiky turtles, and the piranha plant.

The simplest form of adversary is the Goomba. The Goomba only walks horizontally at a slow speed. If a Goomba hits a wall, its direction changes, and it starts to walk in the other direction. Goombas may be killed by jumping on top of it, removing it from the game. If Mario were to walk into one, Mario would die. There are three types of turtles, red, green, and spiky turtles. Each of the three turtles has different movement patterns and game mechanics, actually making them into three very different adversaries. What's important is that red and green turtles introduce an interesting game mechanic that may be used to the player's advantage or demise. If Mario jumps on top of a red or green turtle, the turtle shell is left on the ground. By running into the shell, Mario may use it as a straight projectile to kill all adversaries it collides with. However, if Mario is hit by the fast-moving shell, Mario dies. The last adversary is the Piranha Plant. The piranha plant is an enemy which occasionally protrudes from some pipe obstacles. For a human player, this obstacle is easy to surpass; in most cases, the player can wait for the plant to disappear and then cross the pipe.

An extra layer of complexity is added to the game by enhancing some adversaries. Turtles and goombas may be augmented with wings, which makes their behavior harder to predict. Their movement changes from going back and forth horizontally to jumping around sporadically. Facing a winged enemy may be a hard challenge to overcome, even for an intermediate player.

The third problem faced by a Mario player is obstacles. There are three different forms of obstacles; blocks, pipes, and hills. Block obstacles are often placed together with other blocks in levels of the original Super Mario Bros game. However, the blocks are often not as intelligently placed in procedurally generated levels. By combining several blocks, the player may have to move past stairs, walls, maze-like dead ends, or blocks floating in the air.

Pipes are obstacles similar to wall obstacles as they must be overcome by

jumping. These may be hard to get past if it is enhanced with a Piranha Plant, making it significantly harder to get past. A player may also get stuck if it runs too close to the pipe, as the player cant jump past the rim if it stands too close. Hills are the last obstacle Mario faces. Most Mario levels have a ground on the bottom of the level that the player walks on. However, this ground floor does not have to be horizontal, some levels have ground floors where the height varies.

In conclusion, Mario levels challenge the player with many sub-problems of varying difficulty. Each obstacle and enemy in itself is not very hard to get past, but when many are combined, they may present a grand challenge of great complexity even for an expert-level human player.

2.11.2 Mario AI competition

The Mario AI Championship was a series of competitions for reinforcement learning and procedural content generation [27] [28]. It was hosted yearly between 2009 to 2013, where the years 2009 and 2010 sparked the most enthusiasm in the AI community.

As a result of the competition, there is an extensive framework called the Mario AI Framework. The framework is based upon Infinite Mario Bros, an open-source clone of Super Mario Bros created by Markus Persson.

The Mario AI framework is a community-driven open-source implementation of Super Mario Bros in Java. The framework includes not only the Super Mario Bros game but also different types of level generators and a large library of Mario levels to explore. This makes the framework excellent for training reinforcement learning agents where a large curriculum of varying difficulty or level generation is required.

2.11.3 Challenges for AI

Several features make the Super Mario Bros game particularly interesting from an AI perspective. The most important aspect is the very rich and high-dimensional environment representation. When a player observes the game, it can only see a small part of the current level with the camera following Mario. Even though it is only possible to observe part of the level, the observations include dozens of objects such as enemies, blocks, and collectible items. The environment consists of both static and dynamic parts. The static part, such as pipes, blocks, and coins, are laid out in a 15 by 15 grid, whereas moving objects such as enemies move at pixel resolution.

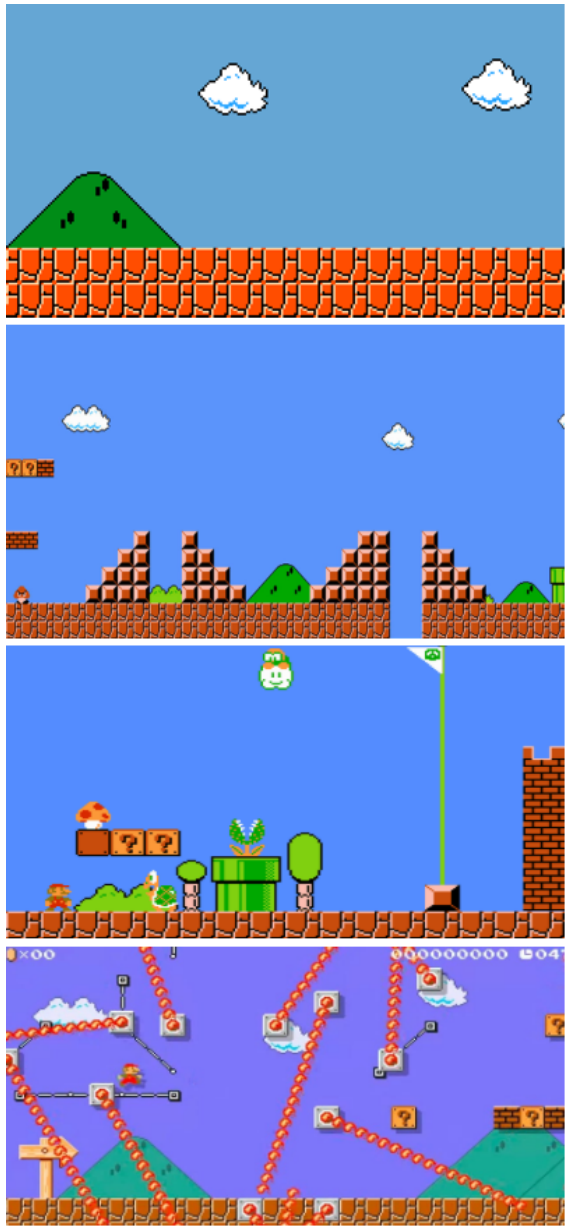

Figure 2.7: Four Mario levels of increasing difficulty are displayed. The first level is very simple, an agent would just have to learn to walk to the right. The second level is of intermediate difficulty, introducing block obstacles, gaps, pipes, and simple enemies. The next level is a hard level, challenging Mario with 3 different enemies in a small space. The last level displayed is of extreme difficulty and would require a Mario player to use planning and timing. For the player to achieve a max score, Mario has to walk from the left on the map to the right and touch the flag, as displayed in the third image. If Mario falls into a gap, touches an adversary, or comes in contact with a flaming rod, the level ends, and the player has to start over.

Chapter 3

State of the Art

3.1 Open-Endedness

The focus of evolutionary computation has traditionally excluded the pursuit of open-endedness, in contrast to the more philosophical questions posed by the artificial life community [23]. Recently a new class of algorithms focusing on abandoning objectives has spurred some interest. Novelty search is one of these approaches which tried to take a step toward open-endedness [15] [16]. By not using an objective function, it instead searches for novel behaviors in an attempt to create a divergent process. The approach did not produce excellent results, as it spent a lot of time searching through uninteresting parts of the solution space. Novelty search is also dependent on creating a measure of novelty that is non-existent in nature. Even though novelty search might be flawed, abandoning the objective function serves as an interesting perspective.

Inspired by novelty search, new methods focusing on using divergent pressure along with objective functions were introduced. Quality diversity was one such method, using a behavioral characterization to measure novelty while at the same time searching for high-quality solutions through an objective function [19]. Similarly, the Multi-dimensional Archive of Phenotypic Elites method has produced valuable results, such as creating robot walking strategies [18]. Common to all of these techniques is that they require an archive of previous behaviors to determine which individuals provide novel discoveries.

In nature, there is no novelty measure or archive. There is only one fundamental constraint; survive long enough to reproduce. Minimal-Criterion Co-Evolution (MCC) introduced the idea of combining the coevolution of two interlocking populations with the use of a minimal-criterion [2]. It was tested by coevolving a population of mazes and a population of agents, gradually shifting

both populations towards greater complexity. The minimal criterion constraining the population being that each maze has to be solved by one agent, and every agent has to solve at least one maze to be part of the problem and solution populations. By enforcing a minimal criterion, a proliferation of diverse problems and solutions of varying complexity was shown to emerge from a single run without using any behavioral description or archive.

In MCC, there is no other requirement for an individual to be allowed to reproduce than satisfying the minimal-criterion. The algorithm does not fall into the traditional competitive or cooperative coevolution categories but is related. Individuals are not rewarded based on the success or failure of others; instead, satisfying the MC is enough to qualify for reproduction. A notable observation is that this means there is no ranking among individuals. Therefore a selection method free of bias is needed. A method from alife using a fixed-size queue in the artificial life world of Chromaria provides such a bias-free selection method [22]. Agents and environments are stored in each its queue, called the parent queue, ordered by the time they were able to satisfy the minimal-criterion. When the queue is full, the oldest individuals are pushed out of the queue. The intention being to shift the populations towards greater complexity as new individuals are added. The queue's agnosticism intended to allow the algorithm keep many divergent paths open simultaneously [2].

3.2 Goal-Switching

An important question is how evolution can explain the speed at which the present complexity of life arose [13]. Current simulations of evolution are known for having problems with scaling to high complexity. Simulations mimic natural evolution by using nature-inspired tools such as replication, mutation, and selection. A logarithmic slowdown typically arises, the simulation of evolution taking longer and longer to increase fitness. The same slowdown is observed in adaption experiments with bacteria when the environment is constant [1].

In nature the environment faced by organisms is continually changing. Previous studies has indicated that temporally varying environments can affect several properties of evolved systems such as robustness, evolvability [7] and modularity [17]. In particular, switching between goals that share common sub-problems has been shown to generate systems of modular structure [11].

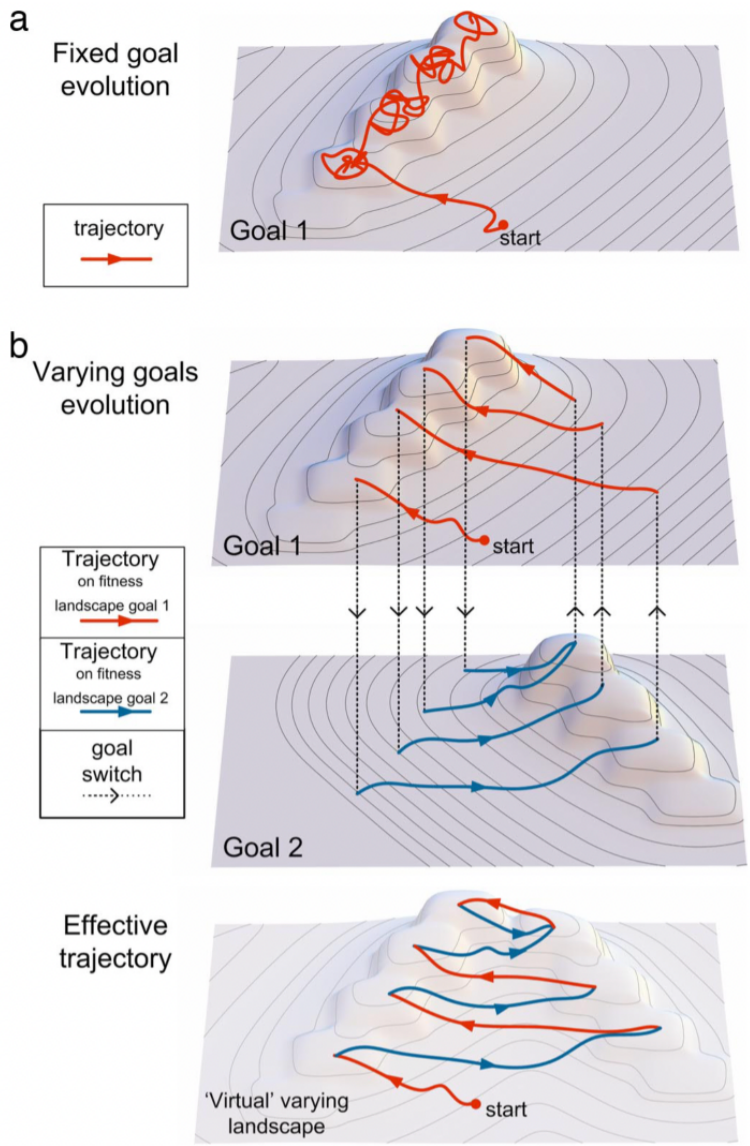

Figure 3.1: Fitness landscape displaying trajectory of fixed goal evolution and evolution using varying goals with common sub-problems. a) When evolution uses a single fixed goal, it tends to get stuck in local optima for a long time. b) Using varying goals during evolution provides a continuous positive gradient by alternating fitness landscape. The population follows a trajectory until it reaches global maxima, which exist in close proximity for both goals. Figure from "Varying environments can speed up evolution" by Kashtan 2007 [12].

By temporally varying goals sharing common sub-problems, the speed of evolution was increased compared to evolution under one fixed goal when using a genetic algorithm for the prediction of RNA structure [12]. Furthermore, it was found that using varying goals with common sub-problems provided the most benefit when the final goal was hard to reach. The findings may suggest that varying goals is key to speed up an evolutionary process, while using goals with common sub-problems may encourage modularity to arise in solutions found.

Goal switching seems to be a powerful tool to speed up an evolutionary process. Temporally switching goals may allow the search to escape local fitness maxima, a common challenge for optimization methods. By progressing towards solving one environment, solutions may experience progress towards solving other environments, at least when they share common sub-problems.

The Paired Open-Ended Trailblazer algorithm (POET) [29] employs goal-switching and minimal-criterions. There are two features of POET functioning as goal-switching mechanisms. First, the transfer of agents between environments switches the goal of the agent that is transferred. The second feature is the mutation of environments. When environments are mutated, a goal-switching mechanism is provided by replacing old environments with mutated environments in the environment population. Agents that are best able to show progression towards solving the newly generated problem are copied and then paired with the new problem. Thus switching the goal for the newly paired agent. One of the goal-switching mechanisms of POET has previously been shown to be essential for the algorithm to solve problems of greater complexity [29]. When transfer was removed, the search process could not solve as challenging problems for the 2D bipedal walker environment.

3.3 Mario

The game Super Mario Bros (SMB) is capable of representing diverse problems of varying difficulty. At the same time as SMB levels can be trivial to solve, generated game levels can be of extreme difficulty. Some levels may even seem impossible to solve for expert human players. Hard levels of the recently released Super Mario Maker game can serve as an example of hard levels created through procedural content generation (PCG).

Level generators for SMB like Ben Weber's award-winning level generator receive a parameter vector consisting of probabilities for game objects and the maximum values of different game features [28]. The parameter vector is essentially an indirect encoding of Mario levels of the same complexity. By combining indirect encodings along with SMB's ability to represent levels of both low and extreme difficulty, Mario provides an excellent playground for reinforcement learning and coevolution.

Chapter 4

Model

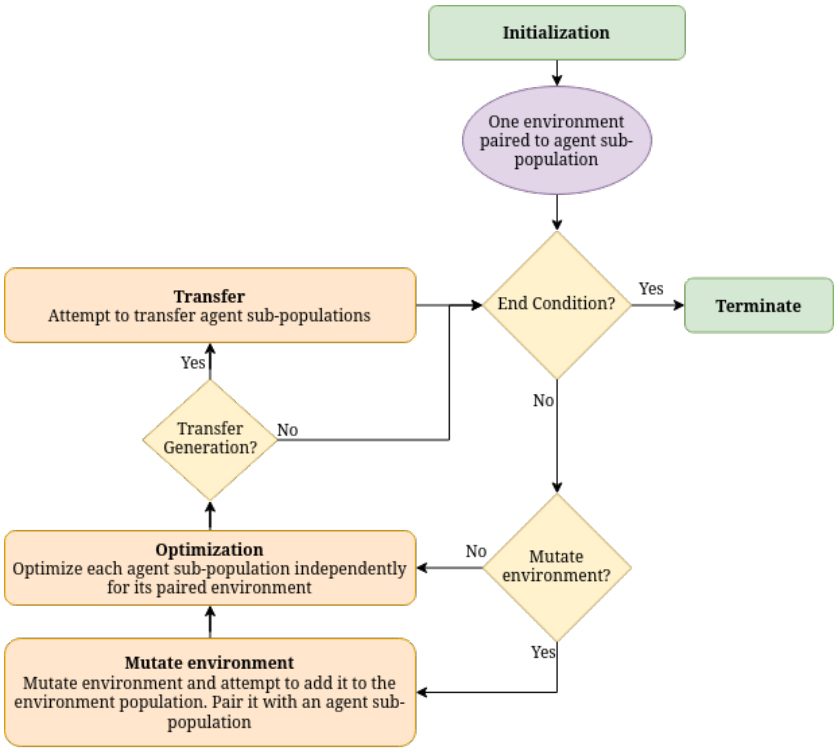

Figure 4.1: Model

4.1 Introduction

The model is inspired by the Paired Open-Ended Trailblazer (POET) algorithm described in section 2.10. For agent optimization, the model uses Neuroevolution of Augmenting Topologies (NEAT) to learn agent policy, further described in section 2.7. The environments are represented by indirect encodings of game levels for Super Mario Bros.

4.2 Agent-Environment pairs

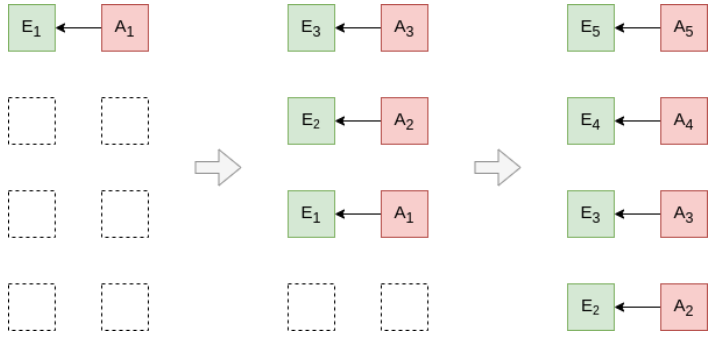

Figure 4.2: Environment queue of size four, each environment paired to an agent sub-population. (1) The algorithm has been initialized, and the environment queue contains a single environment paired to an agent sub-population. (2) Two new environments have been added successfully, but the environment queue is not yet full. (3) Four environments have been added successfully since initialization. Environment E_1 has been pushed out of the queue.

The model maintains a fixed-size queue of environments during evolution. In the queue, the environments are ordered by how long they have been part of the environment population. When the environment queue is full, and a new environment is added, the oldest environment is removed from the population. As new environments are of increased difficulty due to the enforcement of a minimal-criterion, the environment population shifts towards problems of greater complexity.

In the POET algorithm described in section 2.10 each of the environments in the environment queue is paired to a single agent. In this work, every environment is paired with an agent sub-population instead. This is due to using NEAT for optimization instead of evolution strategies. When an agent sub-populations

paired environment is removed from the queue, the sub-population is removed from the agent population. A new agent sub-population takes its place, making the size of the total agent population stay constant. The mutation of environments is further explained in section 4.4.

During evolution, agent sub-populations do not interact directly with other agent sub-populations. Instead, they are optimized for their paired environment independently. However, when transfer attempts are made, agent sub-populations compete for survival. The best agent sub-population replacing the previously paired sub-population.

4.3 Initialization

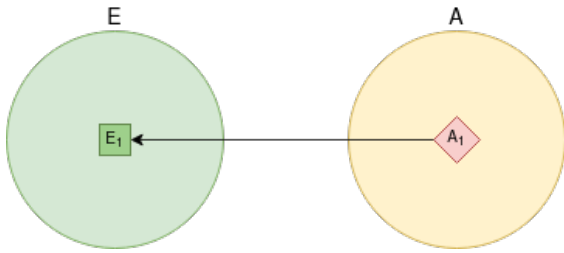

Figure 4.3: environment and agent population after initialization. The environment population contains one environment. The agent population contains one agent sub-population. Environment E_1 is paired to agent sub-population A_1.

To initiate the algorithm, a simple environment is first generated. The initial environment is represented by an indirect encoding which is added to the environment population. All later environments will spring from the initial environment by mutating the environmental encoding.

The second step is to create an initial agent sub-population and pair it with the environment. Each agent is an artificial neural network of minimal topology which is evolved by using NEAT. After the agent sub-population has been paired, it is added to the agent population. More agent sub-populations will later be added as new environments are admitted to the environment population. All agent sub-population have the same size, equal to the total agent population size divided by the length of the environment queue.

After the initialization of agents and environments, the main loop begins.

4.4 Environment Mutation

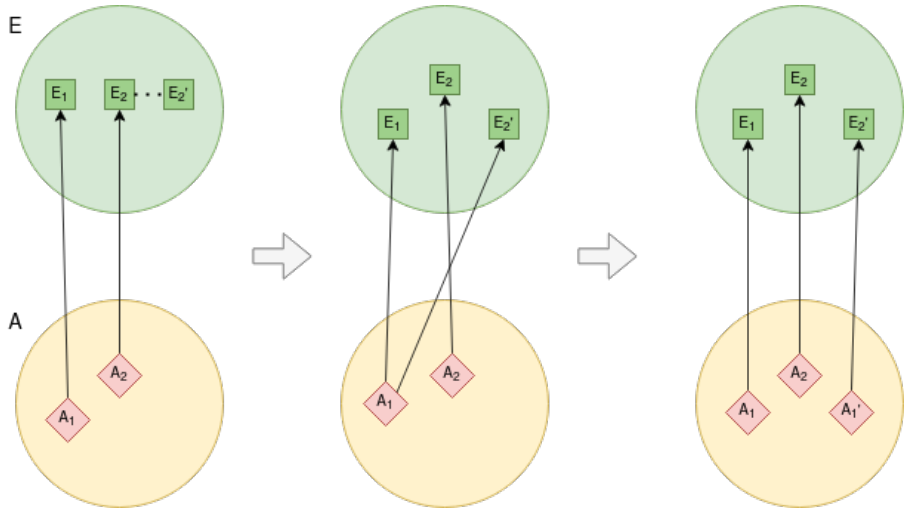

Figure 4.4: Periodically mutated environments are added to the environment population if the agent population is able to satisfy the minimal-criterion. (1) A mutated environment is attempted to be added while the environment queue is not full. The mutated environment E_2' is the child environment of E_2. (2) Agent sub-populations are evaluated in E_2'. A_1 is able to satisfy the minimal-criterion for E_2'. (3) A_1 is duplicated, creating A_1'. Agent sub-population A_1' is paired to E_2'. A_1 and A_1' will diverge as they are optimized for their respective environments.

Every n generations an attempt to mutate and add new environments is performed. If one of the agent sub-populations is able to satisfy a minimal-criterion for the level, it is added to the environment population and paired with the sub-population.

First, an environment is selected from the environment population randomly. Random perturbations of the environmental encoding are then used to generate several candidate environments. Many of these environments are likely too hard or too easy for the agent population, making the agents explore parts of the problem space unlikely to further learning. Therefore a minimal-criterion is enforced. If an agent sub-population is able to show some progress towards solving a candidate environment and the environment is not too easy, the environment is added

to the environment population and paired with the best agent sub-population. To determine if an environment is too easy to be explored, agents must not achieve more than a certain performance threshold. For example, for an obstacle course environment the performance threshold could be that agents should not be able to progress further than half of the new obstacle course.

When an agent sub-population is able to satisfy the minimal-criterion for a new environment it is paired to the environment. The pairing means that the agent sub-population is duplicated and added to the agent population. After the duplication, the sub-populations will diverge as they are optimized for each their respective environment.

Due to the fixed-size queue of the environment population, environments will sometimes be pushed out of the population along with their paired agent sub-population. This occurs when the environment mutation is able to add new environments, and the queue is full. Since the environments are ordered by how long they have been in the population, the oldest environments are pushed out of the queue to make space for the new environments.

4.5 Optimization

Agent sub-populations are optimized independently for their paired environment during every generation.

Each agent sub-population is optimized through evolution. First, every agent is evaluated against its paired environment using a single-objective fitness function. The fitness of each agent being the distance traveled towards the end of the game level. After all agents are evaluated, NEAT is used to select agents for reproduction. When agents have been selected, the genomes are subject to crossover and mutation. Crossover mixes genes of each parent genome to create a child genome. By using NEAT historical markings, a crossover of both neural network parameters and neural network topology of the two parent individuals is performed. The resulting child genomes are then mutated, genes randomly perturbed to introduce variation to the agent population. NEAT also uses speciation to protect innovations, as new topologies are unlikely to be applicable immediately. A further explanation of NEAT is provided in subsection 2.7.

The goal of the optimization step is to make each agent sub-population progress towards solving its paired environment.

4.6 Transfer

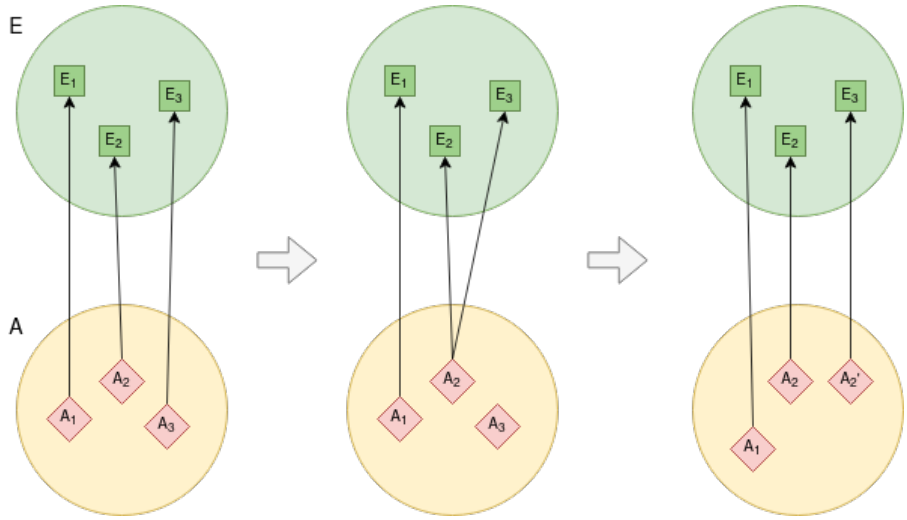

Figure 4.5: Transfer of agent sub-population. (1) Initially, three agent sub-populations are paired to each environment. (2) A transfer attempt is made, and A_2 is determined to show more progress towards solving E_3 than its previously paired agent A_3. (3) A_3 is removed from the agent population, and a copy, A_2', of A_2 is made. A_2' is then paired to E_3.

Every k generations agent sub-populations are attempted to be transferred between environments. When the transfer step is finished, every environment is paired to the agent sub-population showing the most progress towards solving each environment.

To determine which agent sub-population should be transferred, every agent is evaluated against all active environments. The agent sub-population containing the best agent for each environment is then paired with each respective environment. This allows skills that are learned from other environments to provide progress toward solving other environments. If an agent sub-population is able to be transferred, a copy of the agents is made, and they are added to the agent population. The previously paired agents are removed but not necessarily lost immediately, as they might be transferred and paired to another environment.

By allowing skills to be transferred from one environment to another, we might speed up the evolutionary search process. One environment might be used

as a stepping stone toward solving another. Goal switching may also help the agent population get out of local optima for some environments, replacing stuck agent sub-populations.

4.7 Environment

Environments in the environment population are represented by an indirect encoding. The indirect encodings are used to generate the actual environment in which agents are evaluated in. Such an indirect encoding is called an environmental encoding (EE), used to generate the environmental characterization (EC). The encoding may also be interpreted as the environment genotype, indirectly encoding the environment's phenotype. Using an indirect encoding has several practical benefits, such as searching in a smaller problem space and allowing environments to be easily mutated in a coevolutionary setting.

An environmental encoding can, for example, be a parameter vector that defines properties used to generate the environmental characterization. When new environments are added to the environment population, the environmental encoding of the parent environment is mutated. Each EE gene has a probability of being mutated, adding a random perturbation to the allele. By combining mutation of environmental encodings and minimal-criterions, the environmental population will gradually shift towards more complex problems as agents improve their policy.

4.7.1 Mario

We will explore the model through the Super Mario Bros game for this work. Mario provides an excellent test-bed as the Mario AI Framework is implemented in Java and has several level generators used for content generation. Java allows us to parallelize the evaluation of agents easily, utilizing POET's property of scalability through parallelization. The level generators of the Mario AI Framework can be used to generate environments by using a parameter vector that indirectly describes Mario levels.

Level generator

To generate Mario levels, the winning content generator from the 2009 Mario AI competition by Ben Weber is used [28].

The level generator takes in a parameter vector defining probability of game obstacles, enemies, and other game objects. Using the parameter vector, a Mario level is generated.

Parameter	Value
Gap Range	10
Ground Max Height	5
Pipe Minimum Height	2
Coin Height	5
Chance Platform	0.1
Chance End Platform	0.1
Chance Block Coin	0.1
Chance Coin	0.1
Chance Block Power-Up	0.1
Chance End Hill	0.05
Chance Hill Change	0.05
Chance Change Gap	0.1
Chance Block Power-Up	0.1

Table 4.1: Environmental encoding parameters that stay static during the co-evolutionary cycle. The parameters do not affect the difficulty of game levels generated and are not subject to mutation.

Some parameters like obstacle and enemy probability significantly affect the difficulty of generated levels. Other parameters affecting game objects like coins are not as essential and mostly add noise to each generated level. Therefore some parameters were chosen to be static, while parameters assumed to affect difficulty significantly were chosen to be part of the environment genotype.

The environmental encoding used to generate each level is the combination of static level generator parameters defined in table 4.7.1 and the dynamic parameters of the environment genotype defined in section 4.7.1.

Environment genotype

The environment genotype consists of genes corresponding to parameters of the environmental encoding. Parameters that are part of the environment genotype are considered parameters that control the difficulty of generated levels. Each gene corresponds to a vital game property, for example, the maximum height of pipe obstacles or the probability of spawning an adversary on a game tile. Table 4.7.1 defines the genes of the environment genotype, their initial value, mutation step size, and mutation probability of each gene.

Mario levels are desired to be as simple as possible when the algorithm is first initialized. Therefore genes of the initial environment genotype have the value of zero. As the genes control difficulty, the environment characterization generated

Parameter	Initial Value	Mutation	Mutation P
Max Gap Obstacle Count	0	1	0.3
Max Gap Obstacle Length	0	1	0.3
Max Pipe Obstacle Height	0	1	0.3
Max Turtle Enemy Count	0	1	0.3
Chance Winged Enemy	0.0	N(0.01, 0.0025)	0.3
Chance Block Enemy	0.0	N(0.01, 0.0025)	0.3
Chance Hill Enemy	0.0	N(0.01, 0.0025)	0.3
Chance Enemy	0.0	N(0.01, 0.0025)	0.3
Chance Pipe Obstacle	0.0	N(0.01, 0.0025)	0.3
Chance Hill Obstacle	0.0	N(0.01, 0.0025)	0.3
Chance Hill Change	0.0	N(0.01, 0.0025)	0.3

Table 4.2: Environment encoding parameters that are subject to mutation. Each gene in the genotype corresponds to a parameter that controls the difficulty of generated Mario levels. As the value of each gene increases through mutation, the difficulty of the environmental characterization increases. Mutation step size and mutation probability were determined during preliminary testing in section 5.2

from the environmental encoding is of the lowest possible problem complexity.

During some generations new environments are created through mutation as described in section 4.4. When an environment is said to be mutated, it is the environment genotype which is mutated. If a probability gene is chosen for mutation, it is incremented by a value chosen from a Gaussian distribution. The mutation step size and mutation probability of genes are defined in table 4.7.1 and were chosen during preliminary testing in section 5.2. The combination of the static level generator parameters and the mutated environment genotype is then used to generate a new level that is added to the environment population.

4.8 Agent

At each game tick, the agent receives the observable game state as input and then chooses an action in response. The agent can choose to activate five different moves (Jump, Sprint/ Fireball, Left, Right, Down), making up a total of 32 (2^5) possible actions for every time step.

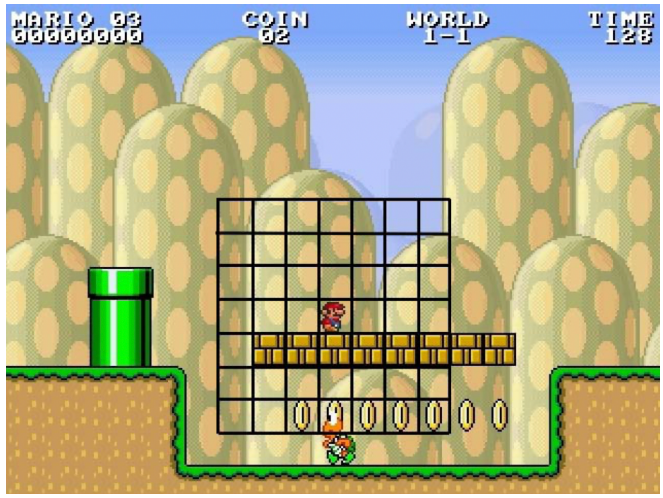

Figure 4.6: The agent observes a seven-by-seven grid centered around the Mario character. Figure from "The 2009 Mario AI Competition" by Togelius [28].

4.8.1 Observation

The observable game state is a seven-by-seven tile grid centered around Mario displayed in figure 4.6. Each grid has a binary value indicating if the grid contains a game object or not. The representation was chosen as it is the previously used game representation for Mario AI competitions [26]. The Mario game is a partially observable Markov decision process, which means that the underlying process is a Markov decision process, but the entire state can not be observed.

4.8.2 Controller

The agents are artificial neural network controllers that determine which action to take for every game state. The artificial neural networks receive a vector of length 41 as input and output a vector of length 5. The output vector determines which action to take.

Hidden nodes of the neural network use sigmoid activation. The output nodes use step activation, as each of the five moves (Jump, Sprint/ Fireball, Left, Right, Down) are binary choices for each game step. Combining the five binary choices makes up a total of 32 possible actions.

Hyperparameter	**Value**
Input Nodes	41
Output Nodes	5
Node activation	Sigmoid function
Output activation	Step function

Table 4.3: The agents are artificial neural network controllers. The network has 41 input nodes and five output nodes. A sigmoid function activates hidden nodes while a step function activates the output nodes.

Chapter 5

Experiments and Results

The experiments in this chapter seek answers to the research questions posed in section 1.1.

Section 5.1 defines initial hyperparameters, the presentation of experiments, and how to interpret their results. Section 5.2 describes the preliminary testing that was performed to test default hyperparameters. Section 5.3 introduces the experiments. The experimental setup is explained in section 5.4. Finally, the experiments and their results are presented in section 5.5 and 5.6.

5.1 Introduction

The following section describes how the experiments are presented and how to interpret them.

5.1.1 Experimental parameters

curriculum	POET generated
stop criterion	1000 generations
runs	50
minimal-criterion	[0.0, 0.1, 0.2]

Table 5.1: Example experiment: Experimental parameters

Experimental parameters of an example experiment is presented in Table 5.1. The value pairs in the Table are experimental parameters describing how the

experiment will be executed. In an experiment, all combinations of experimental parameters will be used.

In the example experiment described in Table 5.1 there are three settings of experimental parameters, each set applying a different minimal-criterion. In this experiment POET generated curriculum will be used with a floor minimal-criterion of 0.0, 0.1, and then 0.2. Each coevolutionary run will last for 1000 generations until terminated and repeated 50 times for each minimal-criterion.

5.1.2 Hyperparameters

Hyperparameter	**Value**
Agent population size	256
Agent sub-population size	64
Environment population size	4
Minimal-Criterion	(0.21, 0.85)
Mutate environment	20 generations
Max levels added	2
Transfer frequency	30 generations

Table 5.2: Default POET hyperparameters

In addition to experimental parameters, there are default hyperparameters for each experiment. The default hyperparameters are determined by adopting knowledge from previous work and through preliminary testing. Results of some experiments are used to alter default hyperparameters. If a default hyperparameter is altered, it will be stated clearly, and the new value will be defined in the subsequent experiment's experimental parameters.

Hyperparameters for the NEAT were taken from the work on MarI/O by Seth Bling, which applied NEAT to the Mario environment. The NEAT hyperparameters are static over all experiments and displayed in Table 5.3.

Default POET hyperparameters are also adopted from previous work in the bipedal walker domain [29]. In the bipedal walker domain, a preference-based fitness function was used, consisting of several objectives. In the Mario environment, the fitness of an agent is decided only by one objective, the distance traveled. In the bipedal walker experiments, the fitness function can take on values between 0 to 230. For an environment to be admitted to the environment population, an agent must achieve fitness between 50 and 200 to satisfy the minimal-criteiron. When mapping the minimal criterion onto our fitness function, we get the minimal criterion (0.21, 0.85), meaning that the agent population

must solve more than 21 percent of the new level and less than 85 percent. The default POET parameters are shown in Table 5.2.

Hyperparameter	**Value**
Input Nodes	41
Output Nodes	5
Node activation	Sigmoid function
Output activation	Step function
Hidden Nodes	10000
Delta Disjoint	2.0
Delta Weights	0.4
Delta Threshold	1.0
Mutate Connections Chance	0.25
Perturb Chance	0.9
Crossover Chance	0.75
Link Mutation Chance	1.0
Node Mutation Chance	0.5
Bias Mutation Chance	0.4
Step Size	0.1
Disable Mutation Chance	0.4
Enable Mutation Chance	0.2
Stale Species	15

Table 5.3: NEAT hyperparameters

5.1.3 Results

Results which are presented in tables are based on the data from the best individuals of the final populations when reaching the stopping criterion. The presented values will be the mean over all runs executed. Values presented in graphs will also be the mean value over all runs, where the number of runs will be presented in the experimental parameters of the experiment. The standard deviation (SD) is also represented along with mean values.

Some experiments are dissimilar in nature, some requiring quantitative analysis and some qualitative. Therefore, in Phase 1 of the experiments, results will be interpreted quantitatively. Phase 2 will, on the other hand, be interpreted both quantitatively and qualitatively.

Some results are presented with p values to show that the results are statistically significant. When there are two or more independent groups, a One-Way

ANOVA test is used to determine significance. If p-value < 0.05, the results are considered statistically significant.

5.2 Preliminary Testing

The model is subject to a large number of parameters. For the POET algorithm, it is essential to find parameters that allow the population of environments to gradually increase in difficulty. If the minimal criterion is too easy to satisfy for the agents, there will likely be less progress toward solving complex problems. On the other hand, if the minimal-criterion is too hard to satisfy, there might be no clear trajectory through the fitness landscape as problems are too hard to learn from. This would occur if, for example, the agent population would not have to show any progress towards solving an environment before admittance to the curriculum. Likewise, if the criterion allows environments that the agent population is already able to solve into the curriculum, the algorithm would be exploring parts of the problem space of little interest.

The default POET hyperparameters found in subsection 5.1.2 were subject to testing before starting the experiments. Most runs of the coevolutionary system were able to run without facing any problems. However, the algorithm occasionally faced premature convergence due to the inability to satisfy the minimal criterion when generating new environments. It was concluded that it was too hard for the agent population to satisfy the floor of the minimal-criterion, 0.21, during the early stages of the coevolutionary algorithm. Therefore it was decided to reduce the floor of the minimal-criterion to 0.1. After these minor adjustments, there was no problem with premature convergence. Final default POET hyperparameters are displayed in Table 5.4.

Hyperparameter	**Value**
Agent population size	256
Agent sub-population size	64
Environment population size	4
Minimal-Criterion	(0.1, 0.85)
Mutate environment	20 generations
Max levels added	2
Transfer frequency	30 generations

Table 5.4: Default POET hyperparameters after preliminary testing

5.3 Experimental Plan

Experiments are divided into two phases. The first phase explores the components and features of the POET algorithm. Importantly it performs an ablation study of the minimal criterion and goal-switching. The second phase explores convergence and artifacts generated by the model when it is allowed to run for a long time. Table 5.5 describes each phase and its experiments, along with the purpose of each phase.

Phase 1: Explore components of the POET algorithm

Experiment 1: Investigate how the amount of environment - agent pairs affects performance. Is there any merit in dividing the agent population into several sub-populations and optimizing them independently?

Experiment 2: Investigate the effect of transfer on the co-evolutionary process. Is the transfer of agent sub-populations necessary?

Experiment 3: Explore how the difficulty of the minimal criterion affects the coevolutionary process. What is the importance of minimal-criterions in an automatic curriculum-building process?

Phase 1 will determine the importance of the minimal-criterion and goal-switching in the POET algorithm. We will also explore whether exploring several environments in parallel improves the search process.

Phase 2: Explore the open-ended properties of the model in the Mario environment

Experiment 4: Explore convergence and artifacts generated by POET. Does the coevolutionary algorithm converge? How complex are the problems that are generated and solved during each run?

Experiment 5: Does late-stage agents forget previously learnt behaviours? How well does the agent population fare when they are introduced to problems the population has previously solved?

Experiment 6: Is NEAT capable of solving complex environments solved by the model through direct optimization? Does POET provide any benefit compared to direct optimization?

Phase 2 explores artifacts created by the POET algorithm to determine the complexity of environments and their solutions. Convergence of the evolutionary search is also studied to see if there are any signs of open-endedness.

Table 5.5: Experimental Plan

5.4 Experimental Setup

All experiments are executed using a single AMD Ryzen 7 5800H processor and 16GB of RAM. Experiments use the default hyperparameters defined in section 5.2 combined with experimental parameters. If the experimental parameters of an experiment overlap with default hyperparameters, the experimental parameters are applied.

A measure of difficulty has been made to measure the agent populations' progression towards solving more complicated problems. The measure is based upon the environmental encodings, where each gene corresponds to either difficulty of game obstacles or the difficulty introduced by adversaries. Four environment genes determine how many objects of certain types are to be spawned or their difficulty, for example, the maximum amount of gaps gene. These genes have larger values and mutation step sizes than genes controlling game object probability. To avoid the genes determining max values overshadowing probability genes, they are downscaled by a factor of 100. The max gap length gene, max turtle count gene, gap length gene, and max pipe height gene are divided by 100 in the difficulty measure. The resulting environment difficulty measure:

$$difficulty = 0.01*maxGapLength+0.01*maxTurtles+0.01*gapLength+0.01*pipeHeight+pWinged+pBlockEnemy+pHillEnemy+pEnemy+pPipe+pHill$$

5.5 Experimental Phase 1

Phase 1 of experiments will determine the importance of minimal-criterions and transfer in the Paired Open-Ended Trailblazer (POET) algorithm. We will also study whether optimizing for several environments in parallel enables the search process to solve problems of greater complexity.

5.5.1 Experiment 1

curriculum	POET generated
stop criterion	1000 generations
runs	10
agent population size	256
environment population size	[1, 2, 4, 8]

Table 5.6: Experiment 1: Experimental parameters

Experiment goal: *Investigate how dividing the agent population into sub-populations affects progression towards solving complex environments. Is there merit to dividing the agent population into several sub-populations and optimizing them independently?*

Introduction

Experimental parameters are presented in Table 5.6. The experiment has four different settings, each using a different environment population size. The maximum size of the environment population dictates the size of agent sub-populations, as the agent population is divided equally between each active environment.

When POET was first introduced, it was applied with massive parallelization over 256 CPU cores, each experiment running for ten days [29]. In this experiment, we investigate how agent populations should be divided into agent sub-populations when working with limited computing resources. The results may be used to choose an agent sub-population size that provides NEAT with enough diversity when combined with POET. It will also determine how many environments should be used for experiments 2, 3, and 4.

One of the experimental parameter settings utilizes an environment queue of length 1. Results of using a single environment can be used to determine whether dividing agent optimization into several smaller parallel search processes is beneficial.

Hypothesis 1 An environment population of size one is expected to get stuck in local optima, hindering progression towards solving complex challenges

With only one environment, transfer between environments will not be possible. As a result, it will be harder for the agent population to escape local fitness optima.

Hypothesis 2 Dividing the agent population into small sub-populations will perform poorly

The largest environment population size will divide the agent population into eight sub-populations of 32 agents. Small agent populations will likely struggle with having enough diversity to progress significantly.

Results

Results are presented in Table 5.7. The results indicate that the agent population is capable of satisfying the minimal-criterion for problems of greater difficulty when there is a moderate number of environments. However, the results are not

Environment population size	Difficulty Mean	Difficulty SD
1	0.24	0.21
2	0.38	0.31
4	0.39	0.18
8	0.21	0.15

Table 5.7: Experiment 1: Results

statistically significant (P = .18). The high standard deviation is likely caused by the use of few runs and occasional premature convergence. It was observed that a few runs are still unable to show any progression, failing to add mutated levels to the environment population from the beginning. If the coevolutionary process was able to show some minor progression towards solving the initial environment, the environment population would start to shift towards more complex problems.

Even though the results of difficulty achieved is not statistically significant, there are clear trends to explore in Figure 5.1, Figure 5.2 and Figure 5.3. Figure 5.1 displays the environment population's ascent towards consisting of problems of greater complexity. When the environment population difficulty increases, it means that the agent population is improving too, as they are able to satisfy the minimal-criterion for problems of increasing complexity.

Using an environment population of sizes 2 and 4 seems to enable the agent population to satisfy the minimal-criterion for the most demanding environments. At first, there seems to be little difference between the two population sizes, one dividing the agent population into two sub-populations of 128, the other four sub-populations of 64. However, there is a clear distinction when the difficulty and levels solved are observed together. Even though the agent populations can satisfy the minimal-criterions for environments of the same difficulty, an environment population of size 4 enabled the algorithm to solve more than 30 problems on average. At the same time, an environment population of size two could only solve five environments. Using an environment queue of size four seems to outperform the other choices significantly if solving levels is desired along with satisfying problems of increased problem difficulty.

Hypothesis 1 states that using an environment population of size one will perform poorly due to being stuck in local fitness optima as transfer is unavailable. Figure 5.1 and Figure 5.2 supports the hypothesis. When we observe the environment population difficulty, we can see that the difficulty slope is less steep than in other groups. This is due to the agent population more rarely being able to satisfy the minimal-criterion for new environments. The evolutionary system is also only able to solve a few environments during each run when a single environment is used. When there is only one environment, it might be replaced before the agents are able to solve the problem, as the environment is frequently replaced. The other factor is that there is no transfer available which might hinder it from escaping fitness plateaus.

Utilizing eight parallel environments allows the search process to solve most levels. However, these problems are much less complicated than the ones solved by the model using four environments. Hypothesis 2 states that dividing the agent population into many small agent sub-populations will perform poorly. Results displayed in Figure 5.1 support the hypothesis, as the runs using an environment population of size 4 achieve a steep environment difficulty slope while simultaneously solving a comparable amount of environments to other groups.

There are several other interesting observations to make from the experiment results. Figure 5.3 presents how often transfers are successful. Transfer does not seem to play an important role, as successful transfers do not occur often. The results suggest that transfer frequency is not the most important POET component. Experiment 2 in subsection 5.5.2 will explore transfer further. Another observation is that environment population difficulty seems to be increasing linearly. This means that the agent population is able to satisfy the minimal-criterion consistently for new environments, even though the difficulty is increasing. One could hope this was an indication of an open-ended process, a more likely explanation being that algorithm has not converged yet. Experiment 4 will explore convergence further in subsection 5.6.1.

A strange observation can be made from Figure 5.1. Something happens to successful transfers around generation 450. Suddenly the successful transfer frequency of environment population of size four decreases. When the run with environment population of size 8 reaches generation 700, the same decrease seems to occur. Looking at Figure 5.1, it seems that the transfer success decreases when the difficulty reaches 0.2, indicating that transfers become harder at a high level of complexity. Even though the transfer success frequency decreases, it does not seem to affect the increase of level complexity. Further indicating that transfer is not the most critical factor of POET, at least when used with NEAT. These results will not be heavily interpreted, but it is a curious finding.

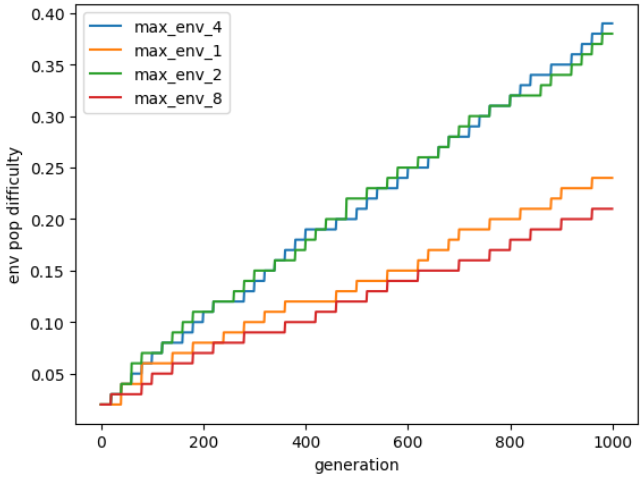

Figure 5.1: Average active environment difficulty

.

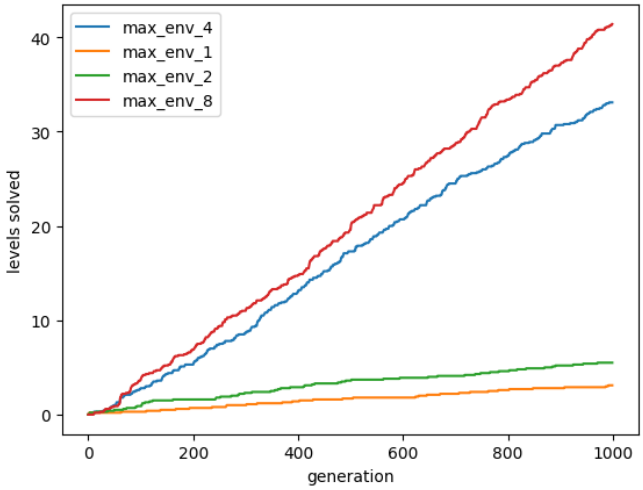

Figure 5.2: Environments solved

.

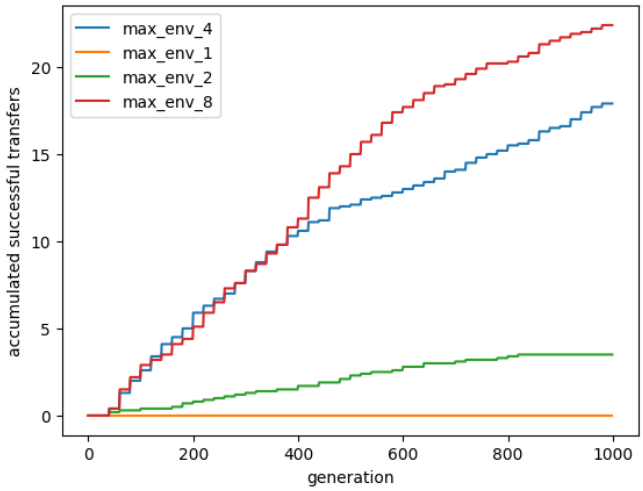

Figure 5.3: Accumulated successful transfers

.

When selecting environment population size for the remaining experiments, the most important measure is the complexity of the environment population. POET aims to drift the two populations towards increasingly hard problems and their solutions in parallel. Figure 5.1 shows that both an environment population of 2 and 4 provides the best drift towards solving complex problems. Therefore one of them should be selected for further exploration. Figure 5.2 provides a valuable result that may be used to determine which of the settings should be used. Four parallel environments create significantly more solutions to problems introduced while exploring problems of the same complexity as two parallel environments. Being able to solve problems is desired, as reinforcement learning attempts to create solutions for problems and not just satisfy a minimal-criterion. Because of this distinction, an environment population of size four is selected for further experiments.

Conclusion

Dividing the agent population into several parallel agent sub-populations which are optimized independently seems to enable the model to satisfy the minimal-criterion and solve environments of the greatest complexity. Dividing the agent population into four agent sub-populations of size 64, independently optimizing each sub-population for their paired environment, was the best division of agents. Findings suggest that transfer might not play a significant part in the coevolutionary process. Instead, it is more likely to be environment mutation and the fixed-size environment queue that drifts the populations towards increased complexity.

5.5.2 Experiment 2

curriculum	POET generated
stop criterion	1000 generations
runs	10
agent population size	256
environment population size	4
agent sub-population size	64
transfer frequency	[0, 5, 10, 20, 40, 80]

Table 5.8: Experiment 2: Experimental parameters

Experiment goal: *Investigate the effect of transfer on the coevolutionary process. Is the transfer of agent sub-populations necessary?*

Introduction
One of the main components of the Paired Open-Ended Trailblazer (POET) algorithm is transfer of agent sub-populations. By occasionally replacing agent sub-populations with a superior sub-population, behaviors learned in one environment allow the agents to progress at solving another environment. This means that an environment may be used as a stepping stone toward solving previously challenging environments.

The question of transfer frequency is a question of exploration versus exploitation. How long should an agent sub-population be optimized for its paired environment before its attempted replaced? This experiment will explore the effect of transfer on the progression towards solving challenging problems. It will also explore whether transfer is necessary for the POET algorithm. The experiment's results will also determine the transfer frequency used in the remaining experiments.

Hypothesis 1 Low transfer frequency is expected to perform slightly worse, as transfer may not be used to avoid local optima as frequently

Although POET has the means of getting out of local optima by goal switching through adding new environments, not having the opportunity to transfer is expected to slow down the search for agents capable of solving complex problems.

Hypothesis 2 Frequently transferring solutions will hinder progression

When transfer is performed frequently, agent sub-populations will be replaced

more frequently. As a result, there will be less diversity in the agent population, as agent sub-populations are not given time to diverge by optimizing independently.

Results

Transfer Frequency	Difficulty Mean	Difficulty SD
0	0.32	0.18
5	0.33	0.21
10	0.21	0.18
20	0.28	0.22
40	0.36	0.18
80	0.25	0.22

Table 5.9: Experiment 2: Results

Table 5.9 presents the average difficulty achieved and standard deviation by transfer frequency. There was no statistically significant difference between the experimental parameter groups ($P = 0.51$). A transfer frequency of 20 means agent sub-populations are transferred every 20 generations.

Figure 5.4 and Figure 5.5 displays the average environment population difficulty and environments solved by generation. Unlike results from subsection 5.6.2, there are no clear differences between trends of the experimental parameter groups. What is striking is their similarity. Transferring agent sub-populations every five generations explores the same environmental complexity as no transfer at all. There is no large difference in the number of environments solved by the two groups. These findings support the findings of experiment 1 in subsection 5.6.2, transfer does not seem to be an essential component of POET. Transfer does at least not play a major role when POET is applied in the Mario environment when agents are optimized by Neuroevolution of Augmenting Topologies (NEAT).

Transfer every ten generations perform considerably worse than transfer every 20 and every five generations. It would be expected to be a correlation between increasing or decreasing transfer frequency and environment population difficulty, but there does not seem to be one. This further supports the hypothesis that transfer does not improve performance considerably. Which setting of transfer frequency performs best seems to be random.

Figure 5.6 shows that frequent transfers increase the accumulated successful transfers. This is expected, as more attempts to transfer are executed. Nevertheless, despite many successful attempts, there is no apparent improvement in environments solved or environment population complexity achieved. The findings further support that transfer does not affect the search process significantly.

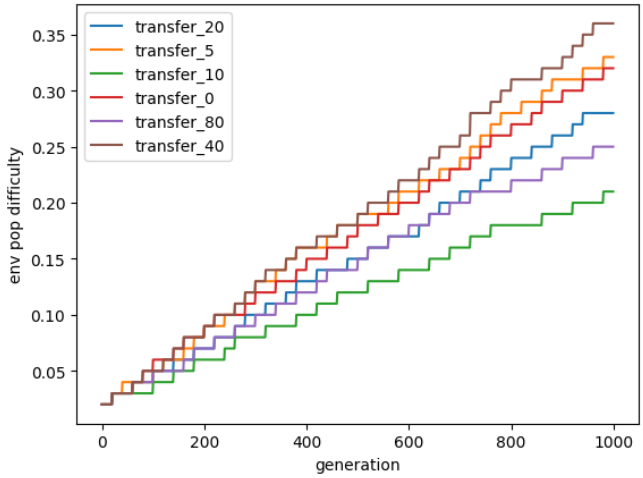

Figure 5.4: Average environment population difficulty

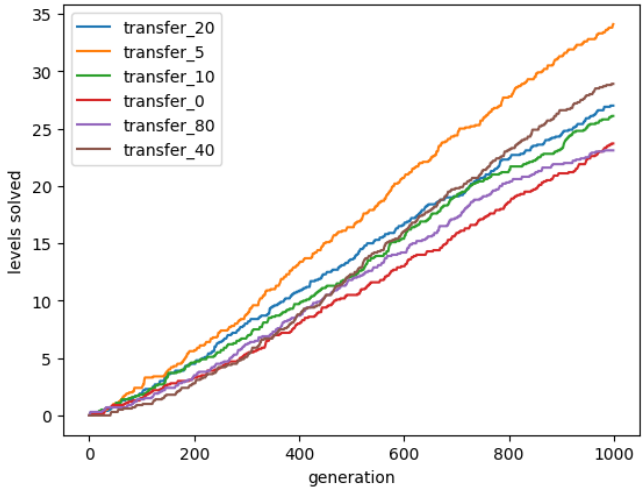

Figure 5.5: Levels solved

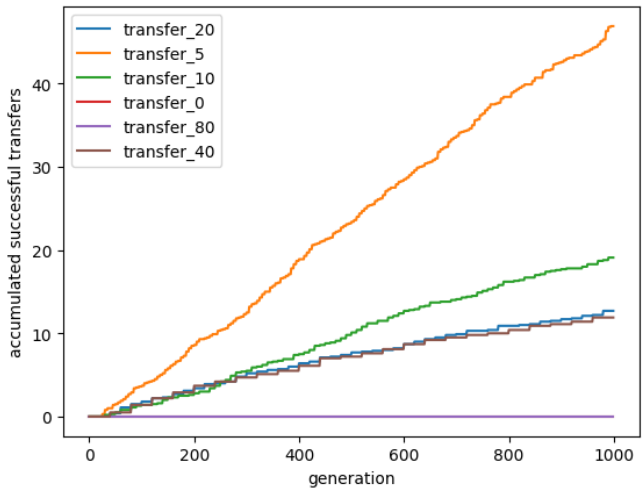

Figure 5.6: Accumulated successful transfers

It should be noted that transfer is an expensive operation, evaluating all sub-populations against all environments to find the agent sub-population most suitable for progressing towards solving each environment. For example, a transfer frequency of 5 would perform $5 * 4 * 256 = 5120$ additional environment evaluations every five generations. This is costly as evaluation often is expensive for reinforcement learning environments.

Hypothesis 1 expected that using a low transfer frequency would perform worse than other transfer frequencies, as transfer could not be used to escape local optima for agent-environment pairs. Figure 5.4 and Table 5.9 disproves the hypothesis. On the contrary, using a low transfer frequency of 1 transfer every 40 generations was able to satisfy minimal-criterions for environment populations of greatest difficulty. Hypothesis 2 stated that using a high frequency would hinder progress toward solving difficult environments. The hypothesis was wrong; using a high transfer frequency performed similarly to other frequencies. Hypothesis 1 and 2 were based on previous work in the 2D bipedal-walker environment, where transfer of agents was found to be of great importance for POET to solve problems of high difficulty [29].

Transferring every 40 generations was able to explore problems of the highest complexity. The results might seem to be due to chance, as there does not seem to be a significant difference between applying different transfer frequencies. However, transferring every 40 generations is selected for further experiments. It was chosen because it achieved the highest environment population difficulty, and the expensive transfer mechanism is rarely used.

Conclusion

Results of the experiment indicate that there is no meaningful distinction between different transfer frequencies when combining POET with NEAT in the Mario environment. Removing transfer enables the agent population to satisfy the minimal-criterion and solve environments of comparable difficulty as runs utilizing the most frequent transfer frequency. The findings support the findings of experiment 1. Transfer does not seem to play a major part in the coevolutionary process.

Transfer has previously been shown to be an essential component of POET to solve complex problems for the 2D bipedal walker problem using Evolutionary Strategies for optimization [29]. The results of this experiment contradicts previous findings, posing the question whether transfer always is necessary.

5.5.3 Experiment 3

curriculum	POET generated
stop criterion	1000 generations
runs	10
agent population size	256
environment population size	4
agent sub-population size	64
transfer frequency	40
minimal-criterion roof	0.85
minimal-criterion floor	[0.0, 0.1, 0.3, 0.5]

Table 5.10: Experiment 3: Experimental parameters

Experiment goal: *Explore how the difficulty of the minimal criterion affects the coevolutionary process. What is the importance of minimal-criterions in an automatic curriculum-building process?*

Introduction
The minimal criterion is an essential component of the model. It is used to decide whether a problem should be admitted into the environment population or not. By making sure that problems admitted into the curriculum are not too easy and not too hard, it presents the agent population with problems that may serve as an effective stepping stone. The intuition is that if a problem is too hard, agents will not be able to progress and gets stuck. On the other hand, if the new environments are too easy, it will not enable further learning as all agents can already solve each problem.

The minimal-criterion has two parts, a floor, and a roof threshold. A roof threshold of 0.85 will be used for all runs, which means that if an agent sub-population is able to progress 85 percent of the distance towards the obstacle course goal, the environment is too easy. We will explore the use of different minimal-criterion floors, which determine how hard a problem can be to be admitted into the environment population. Using a minimal-criterion floor of 0.0 means that agents do not have to show any progression towards solving environments that are added to the environment queue.

In this experiment, we will explore how hard an environment should be to be admitted into the environment population. We will also investigate if a minimal-criterion is necessary to shift the environment population towards consisting of more complex problems.

Hypothesis 1 If all environments are admitted into the population, no matter how hard they are to solve, the coevolutionary process will converge faster

As the environment population will consist of problems that the agent population has shown no progression towards solving, the performance achieved is expected to be stunted. Agents would likely get stuck and presented with challenges that seem impossible to solve. For example, using no criterion may result in agents which have not yet learned to jump being introduced to challenges where the agent must jump over several gaps while simultaneously avoiding adversaries and overcoming obstacles.

Hypothesis 2 A large minimal-criterion is expected to slow down the progression towards solving complex problems

When the minimal-criterion is large, for example, demanding an agent subpopulation to at least solve half of the problem before admitting it into the population, the model will perform poorly. If the floor criterion is too large, simple problems will be explored, and the population will drift more slowly towards increased difficulty. It might also be hard to add new environments to the population, as the mutation has to create an environment of similar difficulty and characteristics as previously presented environments.

Hypothesis 3 The minimal-criterion used in previous experiments is expected to perform well

As the parameter search in preliminary testing and results of previous experiments selected the existing minimal-criterion of 0.1, runs using the minimal-criterion are expected to perform well

Results

Minimal-Criterion	Difficulty Mean	Difficulty SD
0.0	1.14	0.12
0.1	1.09	0.38
0.3	1.01	0.37
0.5	0.73	0.43

Table 5.11: Experiment 3: Results

Table 5.11 presents the resulting mean difficulty and the standard deviation. The results are determined not statistically significant (P=0.14) by using an ANOVA test for the four experimental parameter groups.

Using Table 5.11 there are three main observations to be made. First, there does not seem to be a significant difference in mean difficulty achieved using minimal-criterion values 0.0, 0.1, and 0.3. The second observation is that having no minimal-criterion has a low standard deviation compared to the three other groups. Finally, the third observation is that a minimal-criterion of value 0.5 performs considerably worse than the three other groups, which may confirm hypothesis 2.

Hypothesis 1 proposes that convergence will occur more rapidly when no minimal-criterion is applied. If the experiment finds no rapid convergence, it would question the value of employing a minimal-criterion in the POET algorithm. When inspecting Figure 6.3 and Table 5.11 at first it might seem that removing the minimal-criterion achieves the best results. Upon further inspection of Figure 5.8 we can see that removing the minimal-criterion makes the algorithm converge faster as it cannot solve new environments. It should be kept in mind that when there is no minimal-criterion applied, the difficulty of the environment population will keep increasing linearly infinitely, as all mutated environments are added as long as the environment is not too easy. Since there is no minimal-criterion stopping too hard environments from entering the population, the difficulty keeps increasing even though agents show no progress towards solving each new environment. Agent populations of experimental parameter groups using a minimal-criterion are, on the other hand, able to show some progress towards solving each environment, displaying that the agent population is improving.

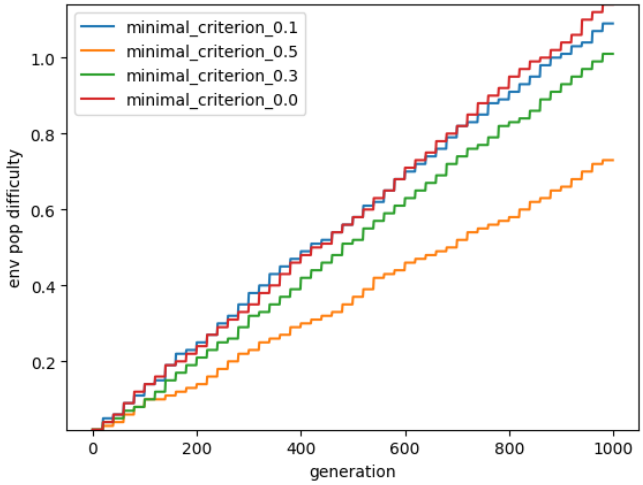

Figure 5.7: Average environment population difficulty

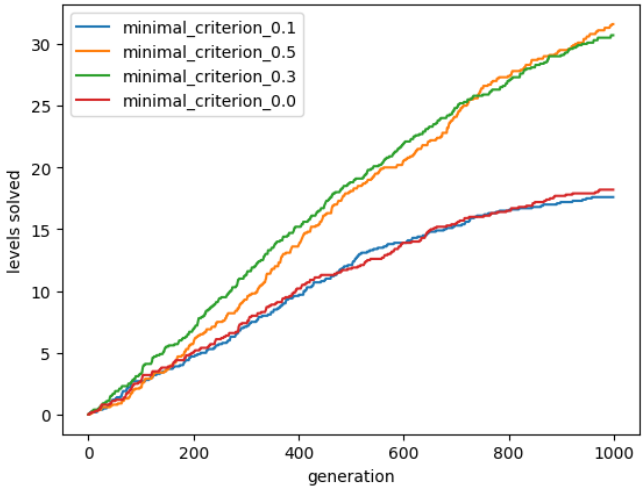

Figure 5.8: Levels solved

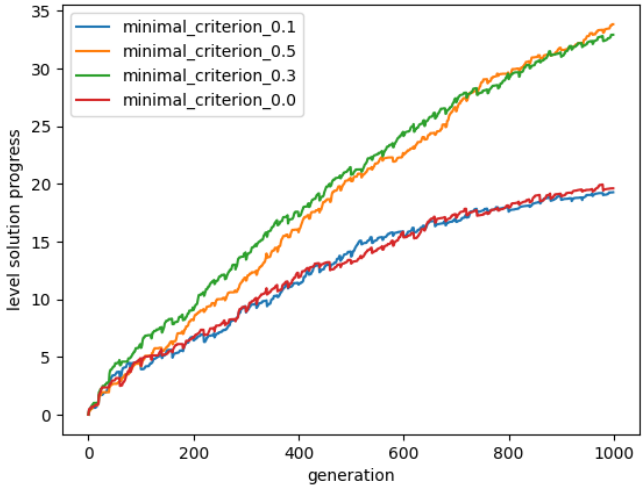

Figure 5.9: Levels solved combined with progress towards solving current environment population

The runs using a minimal-criterion of the value of 0.3 is able to solve new environments at a high difficulty. When the minimal-criterion is non-existent, the algorithm stops showing progress towards solving levels at an earlier generation while exploring levels of comparable difficulty. The convergence shows us that minimal-criterions help POET generate an effective curriculum, allowing the algorithm to solve problems of increased complexity.

Hypothesis 2 is less important but might serve as valuable knowledge for future works. Hypothesis 2 proposes that a too large minimal-criterion will slow the progress towards solving more complex problems. Figure 6.3 has a clear difference between a minimal-criterion of 0.5 and the other hyperparameter settings, confirming the hypothesis. It is a result of mutated environments having to be of low difficulty for the agent sub-populations to be able to solve half of the level. An interesting observation is that using a high floor threshold of 0.5 solves fewer environments than the runs using a threshold of 0.3, even though the lower threshold explores and solves environments of significantly greater difficulty. The results indicate that the criterion has introduced the agent population to a better curriculum, enabling the agent population to solve more challenging problems.

Hypothesis 3 was incorrect. It was expected that the minimal-criterion of 0.1 would achieve good results, as other hyperparameters have been selected while using it. However, figure 5.8 shows that the criterion performs equally to using no criterion. The findings imply that the criterion selected during the preliminary testing enforced a too low threshold on agent environment progression.

Figure 5.9 shows the levels solved combined with the progress towards solving the active environment population. The rugged graph is due to removing old environments in the environment mutation step.

For further experiments, one minimal-criterion differs from the rest. Using a floor minimal-criterion of 0.3 allowed the algorithm to both increase problem complexity, while at the same time solving many of the generated levels. Therefore a minimal-criterion of 0.3 will be used for further experiments in section 5.6.

Conclusion

Removing the minimal-criterion created a worse curriculum than a moderate minimal-criterion, displaying the importance of introducing the agent population to environments of appropriate difficulty. Employing a too high minimal-criterion equally resulted in stunted performance, slowing down the speed of evolution towards solving complex challenges. For further experiments, a minimal-criterion floor of 0.3 was selected, as it enabled the agent population to solve problems of the greatest complexity.

5.6 Experimental Phase 2

Phase 2 explores artifacts created by the POET algorithm to determine the complexity of generated environments and their solutions. Convergence of the evolutionary search is also studied to see if there are any signs of open-endedness.

5.6.1 Experiment 4

curriculum	POET generated
stop criterion	4000 generations
runs	5
agent population size	256
environment population size	4
agent sub-population size	64
transfer frequency	40
minimal-criterion	(0.3, 0.85)

Table 5.12: Experimental parameters

Experiment goal: *Explore convergence and artifacts generated by POET. Does the coevolutionary algorithm converge? How complex are the problems that are generated and solved during each run?*

Introduction
One of the goals of minimal-criterion coevolution and the Paired Open-Ended Trailblazer (POET) algorithm is to move towards an open-ended algorithm. The desire is to create something akin to evolution, seemingly indefinitely creating new problems and solutions of increasing complexity. Therefore it is vital to explore when convergence occurs. Surprisingly results from previous experiments show that the agent population is able to satisfy the minimal-criterion for problems of linearly increasing complexity. One could hope that the combination of gradual topological complexification of neural networks, goal-switching, and the shift towards harder problems resulted in some form of open-endedness. However, the reason is more likely because the coevolutionary algorithm has not yet had time to converge.

This experiment will allow the coevolutionary algorithm to run for a long time. The long run-time will allow us to observe progression towards solving problems of great complexity and explore the levels qualitatively.

Hypothesis 1 The increase in environment difficulty will converge

As the mutated environments become more challenging, it is expected that the agent population will fail to satisfy the minimal-criterion. When the population fails to satisfy the criterion, the increase in environment difficulty will stagnate.

Hypothesis 2 Complexification of neural network topology for the agent controllers will slow down

Hypothesis 3 Late-stage generated levels will be challenging for experienced human players

The experiment is divided into four parts. Part 1 explores the convergence of the coevolutionary algorithm. Part 2 explores neural network controllers and the development of their topology. In part 3, we qualitatively explore levels solved from different generational stages of the coevolutionary runs. Part 4 will let two advanced human players and one intermediate player attempt to solve the most complex environments generated and solved by the model.

Part 1
Figure 5.10 displays average environment population difficulty for each run. After running the algorithm for 4000 generations, the environment population difficulty still increases linearly. These results prove hypothesis 1 false. The results are surprising, as this means the agent population is able to satisfy the minimal-criterion for environments of linearly increasing difficulty. Possibly the process has not yet converged, needing more time to fail to satisfy the 30 percent progression threshold for new mutated environments.

Even though the coevolutionary process has not yet converged with regard to satisfying the minimal-criterion for new environments, it has started to converge with regard to solving environments. Figure 5.11 shows that after about 1000 generations, environments are being solved less frequently. It is a possibility that as environment difficulty increases, the agent sub-populations need more time to optimize for each environment to solve it. Therefore not as many environments are solved, as environments are replaced before the agent population can solve them.

Surprisingly, the environment population difficulty is still increasing linearly. We believe the most likely reason is that more generations are required to observe the convergence of environment complexity. There are other possible reasons, but they do not seem to explain the results. One possible reason might be the algorithm exploiting one of the genes of the environment encoding. Perhaps a mutation of platform probability makes the levels less difficult at late-stage generations, as it might change environment characterization obstacle structure. This would result from wrong assumptions during the selection of genes for the environmental encoding and intricacies of the selected level generator. By qualitative inspection of late-stage solved levels, it does seem that platforms are occasionally being exploited to reduce environment characterization difficulty slightly. For future work, selecting a maximum value for platform probability would be recommended if working with Ben Weber's level generator.

The platform exploit would not be enough to explain why the environment population difficulty is still increasing linearly. The more likely explanation seems to be that the agent population can still satisfy the minimal-criterion for environments of linearly increasing difficulty.

Conclusion

After 4000 generations, the environment population difficulty is still increasing linearly, which means the agent population is able to satisfy the minimal-criterion for increasingly complex problems. We believe there is no convergence observed with regard to environment complexity, as there have not been enough generations for the agent population to fail satisfying the minimal-criterion frequently.

Concerning the convergence of new solved levels displayed in Figure 5.11 we do not deem it to be of considerable importance. Preferably the algorithm would be able to solve levels that are generated at a higher difficulty, but the minimal-criterion has no such demand; it only requires the minimal-criterion to be satisfied for environments of increasing difficulty. As long as the environment population difficulty increases, the agents are improving due to enforcing the minimal-criterion.

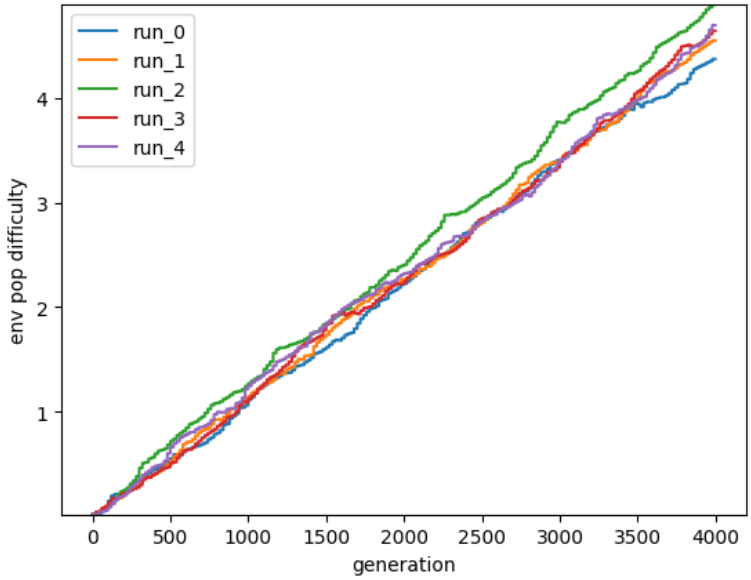

Figure 5.10: Average environment population difficulty

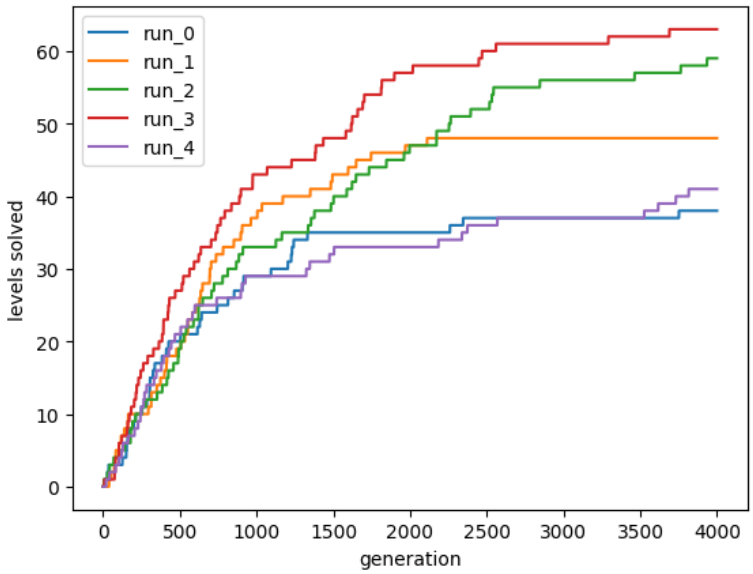

Figure 5.11: Levels solved

Part 2

So far, we have focused on environmental difficulty. The environmental difficulty is a reflection of the agent population's behavioral complexity due to the satisfaction of the minimal-criterion. In this part, we will study the topology of each agent's artificial neural network to explore structural complexity. Figure 5.13 and Figure 5.12 display the average amount of connection genes and amount of nodes in the phenotype of the top performing agent of each agent sub-population. Three runs were selected to be displayed in Figure 5.13 to make the data more easily interpreted and investigate results of particular interest. An entire plot of Figure 5.13 is available in the appendix.

Hypothesis 2 proposes that the growth of neural network topology will slow down during the coevolutionary process. The number of agent connection genes displayed in Figure 5.12 may initially seem to support hypothesis 2, as the genome size increases linearly. However, the genome contains activated and deactivated connection genes, meaning that the phenotype likely has fewer connections and might show tendencies towards convergence. Figure 5.13 displays the average number of nodes in the phenotype of each agent. We can observe that, as opposed to connection genes, the amount of nodes in the phenotype starts to converge, which supports hypothesis 2. Unfortunately, data for the number of edges in the phenotype is unavailable. It would be interesting to see if the number of edges started to converge similarly to the number of nodes.

There are other interesting observations to be made from Figure 5.13. The average number of nodes in the neural network controllers starts with 46 nodes for all runs due to 41 initial input nodes and five output nodes. Hidden nodes are incrementally added to the topology, increasing structural complexity. The average number of hidden nodes seems to trend towards 20, resulting in networks with a total of 66 nodes. Even though the number of hidden nodes stops increasing, the agents continue to satisfy the minimal-criterion for challenges of increasing difficulty, which likely means that the behavioral complexity is increasing.

Figure 5.13 displays results from three different co-evolutionary runs. The first run increases the number of hidden nodes until generation 1500, where it starts to remove hidden nodes, which displays how NEAT attempts to discover minimal network structures. The second run has some curious peaks after generation 2000. We believe these to be a result of NEAT's use of speciation. There is a species in the agent population with a much larger number of hidden nodes than the rest of the species. Occasionally the species become best at solving one of the environments and cause a large increase in the average amount of nodes. These findings show us the value of protecting innovation through speciation.

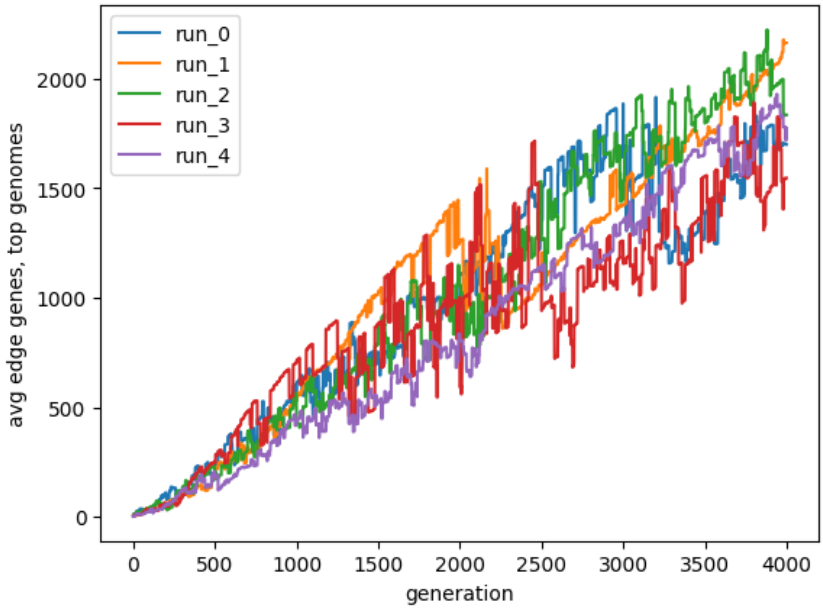

Figure 5.12: Average connection gene count for each agent

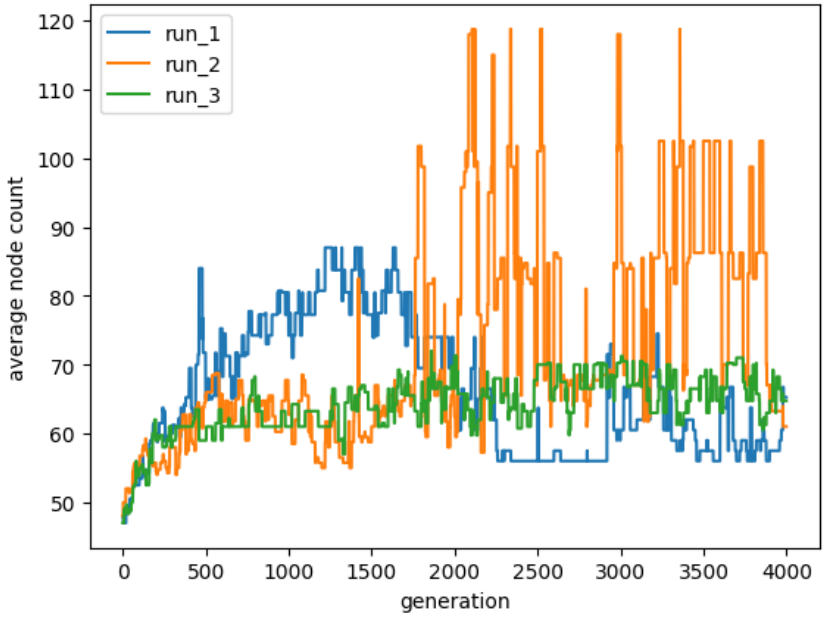

Figure 5.13: Average node count for neural network controllers of each agent

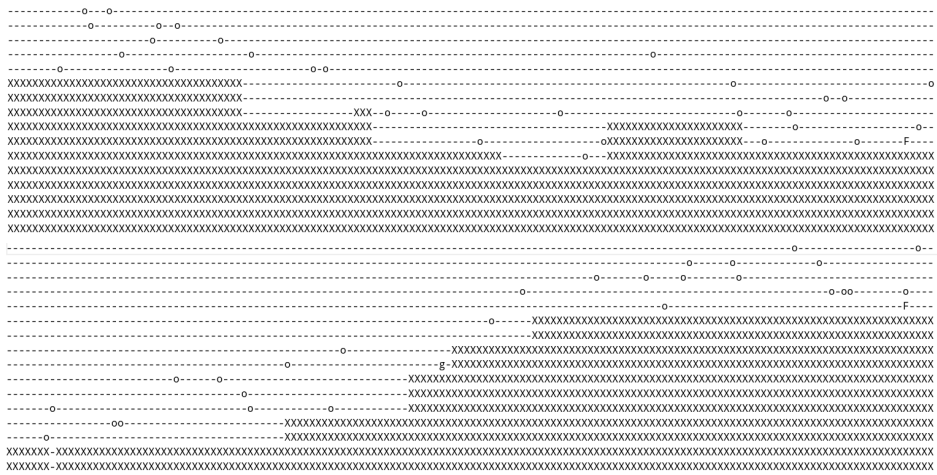

Figure 5.14: Environmental characterization of two levels generated and solved during the first 100 generations

Part 3
This part will explore some of the levels solved through qualitative analysis. We will also try to determine level complexity through gameplay by experienced human players. Table 5.13 can be used to interpret the environmental characterizations displayed in Figure 5.14, Figure 5.15 and Figure 5.16.

Game object	**Value**
Ground	X
Air	-
Platform	%
Coin	o
Pipe	t
Adversary	g, k, r, y
Winged Adversary	G, K, R, Y
Goal	F

Table 5.13: Game objects of the environmental characterization

During the run of the coevolutionary algorithm, diverse challenges of increasing difficulty are generated and solved. Previous experiments have shown that the environment complexity is increasing according to the difficulty measure introduced in section 5.4. To confirm that the difficulty of the environmental characterization is increasing, and not just the measured difficulty, we will explore some of

the solved levels.

Levels generated from the first 100 generations of all five runs were explored. Two were selected and displayed in Figure 5.14. Levels from early-stage generations were all found to be of similar difficulty and complexity. While each level is simple to solve, they are all composed of different sub-problems.

The levels displayed in Figure 5.14 serve as a good example. The first level challenges the player with hills and one block obstacle. Many coins are spawned in the space above the ground, serving as noise that does not significantly affect level difficulty. The second level is a little more challenging than the first, introducing a single simple adversary and a gap. It seems that the levels generated and introduced into the environment population provide diverse challenges of comparable difficulty during the first 100 generations. The comparable difficulty results from the minimal-criterion, only allowing challenges of suitable difficulty into the curriculum. The diversity of the challenges is due to the population of environments, keeping several curriculum paths open for exploration.

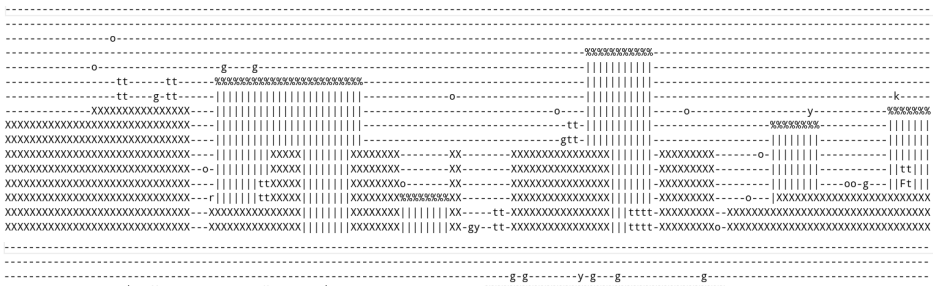

Figure 5.15: Environmental characterization of two levels generated and solved during the first 1000 generations. The first level consists of many challenging gaps and obstacles, while the second level contains many adversaries and some obstacles.

Levels displayed in Figure 5.15 were generated and solved during the first 1000 generations. The two levels pose very different challenges to the player. The first level consists of many obstacles and difficult gaps to cross, while the second level is owing its difficulty to a large number of adversaries. To a human player, the levels would provide a significant challenge, as the two levels are much harder

than the most difficult levels of the original Mario game. From a procedural content generation perspective, the levels are both unique and would provide the player with interesting challenges to overcome.

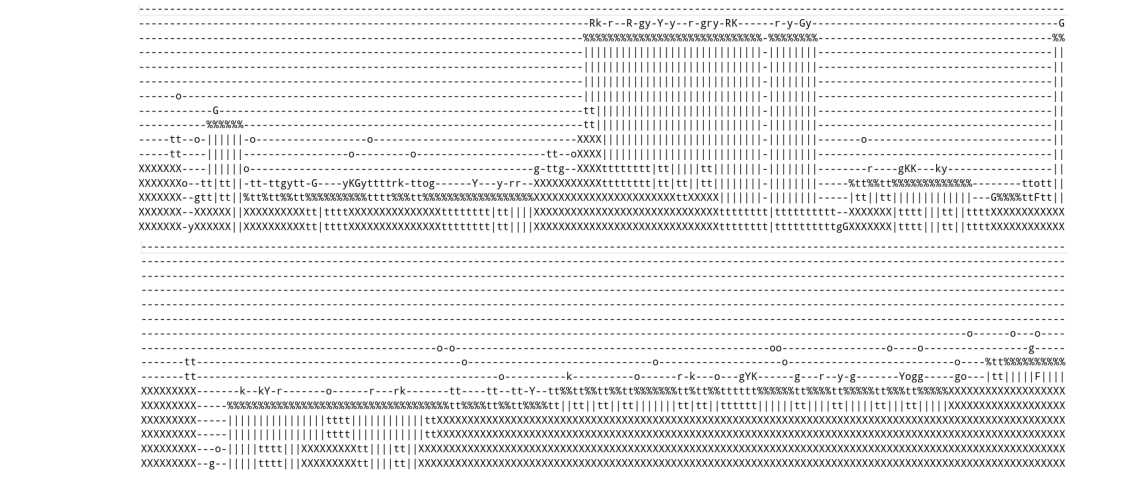

Figure 5.16: Environmental characterization of two levels generated and solved after 2000 generations.

Lastly, we explore some levels solved by the model after 2000 generations. The levels of Figure 5.16 would challenge any human player. We tested the levels on two advanced human players, none of them were able to solve the levels immediately. After several attempts one of the advanced players was able to solve the first level. The second player described the levels as too hard and chaotic due to the extremely large amount of enhanced and regular adversaries.

If the levels solved by the model were to be used for human play, the levels with a lower difficulty might provide more exciting gameplay than the levels solved after 2000 generations. Obstacles such as gaps are not as common in generated levels anymore, as gaps and obstacles seem to be mastered during the first 1000 generations. Thus, new levels added to the environment population have few gaps and obstacles because they no longer provide the agent population with a challenge. Actually, gaps might make the levels easier to solve for the agent, as there are fewer adversaries spawned due to the gaps. As a result, levels contain an extreme amount of adversaries, which only seem solvable by employing strategies to lure out adversaries or use non-human precision. Comparing the solved levels to hard levels of the original Super Mario Bros game, each tile was about five times more likely to spawn an adversary. The adversaries spawned were also much more likely to be enhanced by wings, making the adversaries harder to get past.

While exploring the environments solved after 2000 generations, a few levels were surprisingly found to be deceptive by having dead-ends. However, such levels seemed rare and were likely generated by the level generator and solved by the agents by chance.

One of the reasons the first level of Figure 5.16 was chosen to be displayed is because it contains the platform exploit mentioned in part 1 of this experiment. In the middle of the level, there is a platform on top of the map. All enemies in the area are spawned on top of the platform, which reduces the difficulty. However, having platform probability as part of the environmental encodings is not a big problem due to the minimal-criterion. If levels are too easy to solve due to, for example, too many platforms, the level would be discarded and would never enter the curriculum. Thus, showing how a minimal-criterion can be used to ensure that only problems of interest are explored and why minimal-criterions might be used as an effective tool for automatic curriculum building in machine learning.

The complexity of the game levels increases along with difficulty measured, suggesting that the difficulty measure is accurate. Levels produced by the environmental encodings of comparable measured difficulty are diverse and consist of different combinations of sub-problems. Some are challenging due to obstacles and adversaries. Others are challenging due to many adversaries concentrated in small areas. Even though it does not seem to be a problem yet, the expressiveness of the level generator bounds the complexity of generated game levels. At some point, the levels generated can not contain any more adversaries and there is a limit to how difficult obstacle combinations the generator can create. If open-endedness were the goal, there is a need for unbound possibilities of problems of increasing complexity. Mario is a game where levels can be highly challenging. If the space of all Mario problems were explored, there could arise levels with more maze-like characteristics and levels which require high-level planning. However, if the agents continue to learn, the expressiveness of the procedural content generator will likely become a problem.

Conclusion

Levels solved by the coevolutionary algorithm from different stages were explored. The levels were found to be diverse and of increasing complexity. Levels of the same measured difficulty were unique and consisted of different combinations of sub-problems. Environments generated and solved after the first 2000 generations were found to be of considerably greater difficulty than the most difficult levels of the original Super Mario Bros game.

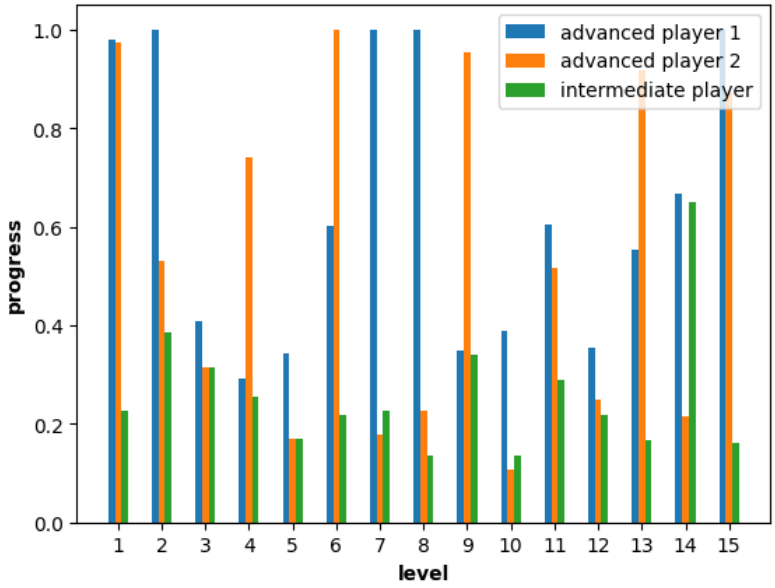

Figure 5.17: Three players were challenged with fifteen levels solved by the model after 2000 generations. All players were given three attempts at solving each level. The progress displayed is the best performance achieved by each player. Only five of the fifteen levels were solved by the players.

Part 4

From each of the five runs of Part 1, three levels that were solved after 2000 generations were selected. Two advanced players and one intermediate player were then used to determine the problem difficulty of the fifteen levels. Each player was given three attempts at solving each level.

Figure 5.17 displays the results of the game-play. The players solved only five of the fifteen levels, confirming that the solved environments are highly challenging. The players fail to show progress at solving some of the levels, barely satisfying what would be the minimal-crtierion for the agent population.

An interesting observation is that the intermediate player seems to fail close to the minimal-criterion for all levels. The second advanced player also fails to progress further than the criterion for some levels. Levels generated and admitted to the curriculum may slightly increase in difficulty after the criterion of 0.3, due to the selection process of candidate environments.

From a content generation perspective, the players were engaged and found

the levels exciting and unique. The players were surprised by the difficulty posed, as there was an extreme amount of adversaries compared to traditional Mario levels. The combination of winged enemies, walking enemies, and projectiles gave the levels slightly maze-like characteristics, requiring human players to plan and time their actions carefully. The two advanced players started using a waiting strategy to lure out adversaries. The agents are unlikely to have discovered such an advanced trick, but for the human players the levels seemed impossible to solve without using it.

Conclusion of experiment

A long run was carried out to explore convergence and artifacts generated by the coevolutionary algorithm. The algorithm was found to not converge with regard to the environment population difficulty achieved, which is used to estimate environment complexity. As the environment difficulty increases, it means that the agent population is able to satisfy the minimal-criterion for increasingly hard problems. However, the coevolutionary algorithm converges with regard to solving new environments.

The neural network topology of agent controllers was explored. It was found that the number of hidden nodes for each neural network trended toward 23 hidden nodes while the connection genes grew linearly. Some of NEAT's components, for example, speciation and minimal network structure, are reflected in the results.

Levels generated by the coevolutionary algorithm were then explored. The levels produced were, as expected, found to be of incrementally increasing complexity but also great diversity. We then used levels from late-stage generations of the runs to challenge two advanced players and one intermediate player. Combined, they were only able to solve five of the fifteen levels, displaying that the environments solved by the model are highly difficult.

5.6.2 Experiment 5

curriculum	POET generated
stop criterion	4000 generations
runs	10
agent population size	256
environment population size	4
agent sub-population size	64
transfer frequency	40
minimal-criterion	(0.3, 0.85)

Table 5.14: Experiment 5: Experimental parameters

Experiment goal:
Does late-stage agents forget previously learned behaviors? How well does the agent population fare when they are introduced to problems the population has previously solved?

Introduction
Starting from minimal neural networks, NEAT gradually complexifies network topology to create minimal neural network solutions. The feature may allow us to discover a topology that is good at solving Mario problems. Furthermore, small networks are less likely to overfit and may incentivize learning behaviors that are helpful towards solving all Mario problems.

In this experiment, all agents of the last generation are evaluated against Mario levels solved during the coevolutionary run. The results will indicate whether the agent learns general behaviors and to what degree it forgets previously learned behaviors.

Hypothesis 1 The agents will show the most progress towards solving early generated levels due to low difficulty

Results
Figure 5.18 shows the average progression towards solving previously solved levels by the final agent population. The agent population is, on average, unable to solve any of the levels, as knowledge likely is forgotten. The phenomenon is a known problem in machine learning, called catastrophic forgetting. Single agents might be able to solve some environments, but the average agent progress is displayed. We can observe that the agent population shows some progress towards solving each level, suggesting the agents learn some behaviors that help solve all levels.

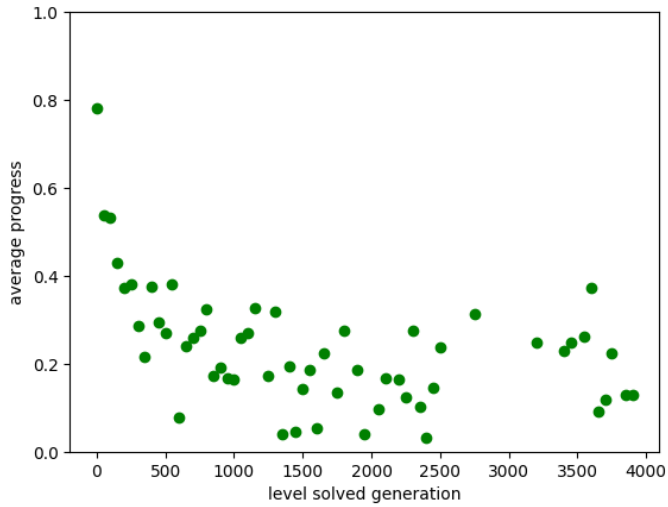

Figure 5.18: progress towards solving previously solved levels by the final agent population. The agent population shows the most progress toward solving early-stage levels. After 750 generations, the population displays similar progress towards solving each level, even though levels are of vastly different difficulty.

The first levels generated and solved are shown the most progress towards solving, confirming hypothesis 1. The result was expected due to the low difficulty of the levels. Interestingly the agent population shows similar progress towards solving levels from all generations after 750 generations. First, this might seem to result from the levels being of the same difficulty. However, findings from experiment 4 in section 5.6.1 determine that levels solved from different stages are of vastly different difficulty.

Conclusion

The agent population retains some previously acquired knowledge, suggesting that some behaviors learned are helpful towards solving all Mario problems. However, the agent population at large is far from solving previously solved levels. Some agents in the population may be able to solve previously solved environments, but the average performance was observed.

optimization	NEAT direct
curriculum	high difficulty level
level count	15
stop criterion	2500 generations
runs	1
agent population size	256

Table 5.15: Experiment 6: Experimental parameters

5.6.3 Experiment 6

Experiment goal: *Is NEAT capable of solving hard environments solved by the model through direct optimization? Does POET provide any benefit compared to direct optimization?*

Introduction
In this experiment, we will determine if POET allowed the search process to solve Mario levels of greater problem complexity than NEAT is capable of solving alone. Fifteen of the most challenging levels solved by the model are selected randomly from a run of experiment 4. All levels were solved between generation 2000 and generation 2500.

We use NEAT to optimize agents utilizing a population of 256 individuals over 2500 generations for each level independently. The results will indicate whether POET's goal-switching and minimal-criterion mechanisms have allowed the agents to solve problems of greater complexity.

Hypothesis 1 NEAT will not be able to solve all challenging problems solved by the model

The curriculum building of POET is expected to have allowed the agent population to use levels of increasing difficulty as stepping stones, allowing it to solve more complicated problems by avoiding local optima.

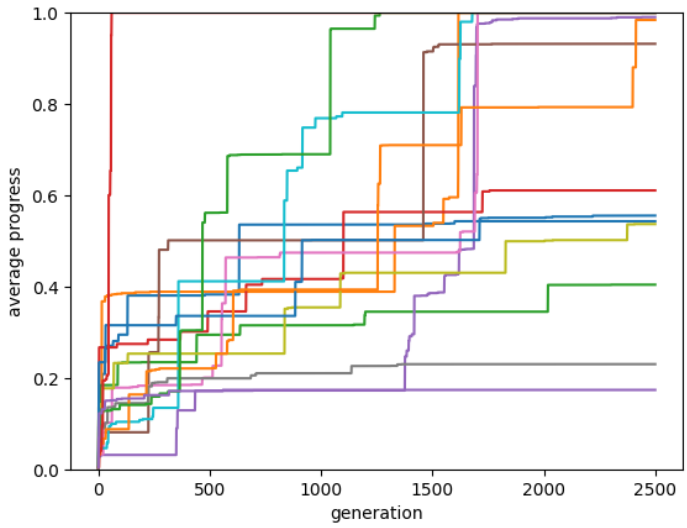

Figure 5.19: Agent population progress towards solving each of the fifteen levels. Each line represents 2500 generations of NEAT optimization for a single environment.

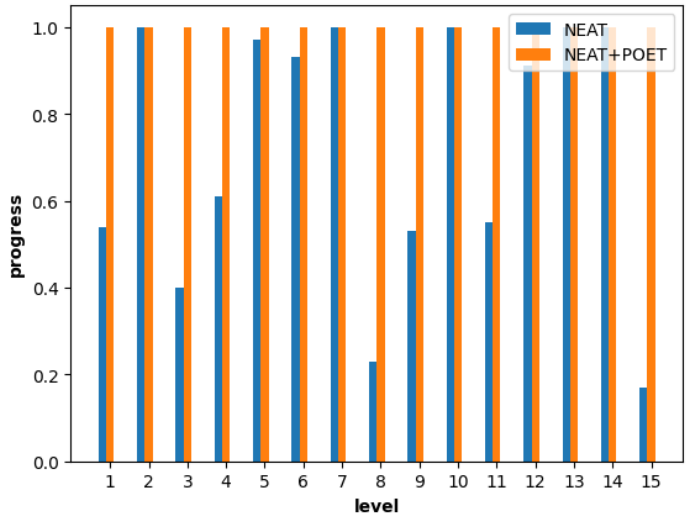

Figure 5.20: All fifteen levels were solved by POET combined with NEAT after a total of 2500 generations. The progress displayed by NEAT is over fifteen runs, one run used for each level. The fifteen runs make up a total of 37500 ($15 * 2500$) generations. Through direct optimization, NEAT was able to solve five of the fifteen levels.

Results

Figure 5.20 shows us how many environments NEAT was able to solve through direct optimization. NEAT solved five of the fifteen levels by using 2500 generations for each independent problem. Figure 5.19 shows that NEAT is close to solving three more problems but is far from solving the remaining seven problems. We can also observe that direct optimization spends much time in local optima, often stuck for many generations. At these fitness plateaus is where goal-switching may be a helpful tool.

After 2500 generations NEAT is sometimes able to solve one complex problem. During the same 2500 generations, the combination of NEAT and POET solved fifteen problems of comparable difficulty. Before solving the fifteen environments, the model generated a diverse curriculum of increasing complexity and solved many of the environments generated. The resulting agent population is also likely to have more robust and modular properties due to goal-switching. These findings suggest that using goal-switching and minimal-criterions for coevolution can work as an effective automatic curriculum builder for reinforcement learning.

Conclusion

Fifteen challenging levels solved by the model were selected for exploration by NEAT. Through direct optimization, NEAT solved five of the environments by using 2500 generations for each independent problem. However, using POET for curriculum building seems to be a fruitful endeavor. After the same number of generations, the combination of POET and NEAT solved at least fifteen problems of comparable difficulty. At the same time, the model builds a curriculum of incrementally increasing complexity while solving many of the problems generated.

Chapter 6

Conclusion

The research goal, research questions, and results are evaluated and discussed in section 6.1. The contributions of the thesis are revisited in section 6.2. Finally, future work is proposed in section 6.3.

6.1 Results and Discussion

Goal *Explore the Paired Open-Ended Trailblazer algorithm*

The master's thesis has explored the Paired Open-Ended Trailblazer algorithm from different angles. In the first experimental phase, we studied essential components of the algorithm to determine how they affect the coevolutionary process. Optimization of agent sub-populations in parallel, the minimal-criterion, and transfer frequency was emphasized. An ablation study was executed for each component to investigate whether they are necessary parts of the algorithm. During the ablation study, we also explored how the components affected the model's performance.

In the second phase of experiments, open-ended properties of POET are explored. The algorithm was allowed to run for a long time, observing convergence of environment population difficulty and environments solved by the agent population. During each run, we focused on the difficulty of the environment population, as it reflects agent behavioral complexity when a minimal criterion is enforced. While POET runs, environments of increasing difficulty and their agent solutions are generated. The artifacts generated from the coevolutionary process were studied to learn more about their properties. Solved environments were explored using qualitative analysis, human gameplay, and direct optimization to determine their complexity and characteristics.

Experiments were conducted while utilizing limited computing resources. Previous works with POET have utilized large amounts of parallel CPU cores. In this work, we explore the use of POET while employing a smaller environment and agent population. The knowledge accumulated may be used to guide future work where large-scale distributed computing is not available.

Research question 1 *How important is the minimal-criterion in the Paired Open-Ended Trailblazer algorithm?*

The importance of the minimal-criterion was explored in subsection 5.5.3. An experiment was conducted to determine how difficult an environment should be to enter the environment population. The effect of removing the minimal-criterion was also studied.

Experiments concluded that removing the minimal criterion caused the coevolutionary algorithm to converge faster, as when the criterion was removed, the agent population displayed less progress towards solving new environments. By using a moderate minimal criterion, the agent population was able to show the most progress toward solving complex problems. The results indicate that the minimal criterion is an essential component of the POET algorithm.

Even though the minimal criterion was determined to be an essential part of the algorithm, enforcing a too large minimal criterion had a negative impact. Only environments posing a minor challenge to the agent population are admitted into the curriculum when the threshold is too large. As a result, the agent population solved fewer environments while the environment population difficulty increased more slowly. The results show that the minimal criterion should be selected carefully as it significantly impacts the coevolutionary process.

Research question 2 *How important is the transfer component of the Paired Open-Ended Trailblazer algorithm?*

Transfer has previously been found to enable POET to solve harder problems [29]. Experiments in subsection 5.6.2 and 5.5.2 suggests otherwise. Transfer does not seem to play a major role, as transfer frequencies were found not to affect results achieved.

There are several reasons transfer might have been found to be of little importance. First, it was observed that the transfer mechanism rarely succeeded, indicating that behaviors learned in one environment are not easily transferable to another. The second reason is the small environment population utilized. In this work, environments are more frequently replaced as the environment population queue is smaller than in previous works. As goal-switching is provided through adding new environments, it might have reduced the importance of transfer.

We conclude that transfer is not always essential to the POET algorithm. Sometimes the transfer mechanism is only functioning as a computationally expensive superfluous operation. Even though transfer is rarely used, the process does not get stuck in local-optima, and environment population complexity increases while new problems are solved.

Research question 3 *Is the Paired Open-Ended Trailblazer algorithm combined with Neuroevolution of Augmenting Topologies for agent optimization able to solve Super Mario Bros levels of high difficulty?*

Based on results from qualitative analysis of game levels, the difficulty measure, and human gameplay, we conclude that POET combined with NEAT can generate and solve Super Mario Bros levels of very high difficulty.

A Mario level of high complexity is a problem consisting of several obstacles combined with many adversaries. A problem of such difficulty is likely to provide a challenge for any human player. Experiment 4 in subsection 5.6.1 explored levels generated and solved by long runs of the POET algorithm. To determine the complexity of solved environments, they were explored qualitatively and by human players of different skill levels.

The problems solved by POET were determined to be of increasing complexity and great diversity. Early stage solved levels consisted of few obstacles and adversaries, providing the agent with simple problems. On the other hand, late-stage levels had extreme amounts of adversaries while at the same time introducing the agent to obstacles. Challenging levels solved by POET were found to be significantly more complex problems than levels of the original Mario game. For example, each game tile was found to be about five times more probable to spawn an adversary than the most challenging levels of the original game. The adversaries were also much more likely to be enhanced, making them more unpredictable.

It should also be noted that the levels explored in experiment 4 are only those solved by POET. The levels of the environment population during late stage generations are of much greater difficulty. Even though the agent population solves few problems at this stage, the agents show considerable progress towards solving them as they satisfy the minimal-criterion for each environment.

After qualitatively exploring game levels, experienced human players were used to determine the difficulty of solved problems. Three players of different skill levels were used, two advanced players and one intermediate player. The problems solved by POET proved to be very hard, challenging the players with too many adversaries while requiring the player to overcome difficult obstacles. The best advanced player could only solve four of the fifteen challenging problems solved by POET. We conclude that the model is able to solve Super Mario Bros levels of considerable difficulty.

Research question 4 *Does the Paired Open-Ended Trailblazer algorithm facilitate learning better than direct optimization for reinforcement learning?*

Neuroevolution of Augmenting Topologies (NEAT) was able to solve one-third of the complex problems solved by POET through direct optimization. The results suggest that the automatic curriculum-building abilities of POET enable agents to learn more complex behaviors.

Experiment 6 in subsection 5.6.3 uses NEAT to optimize agents directly for challenging problems solved by our model. After 2500 generations, NEAT was sometimes able to solve one of the hard problems. After the same number of generations, the combination of POET and NEAT solved at least fifteen problems of comparable difficulty. At the same time, the model builds a curriculum of increasing difficulty while solving many of the environments generated. It should be noted that the model uses some extra evaluations in the simulated environment for goal-switching components.

The research question is intended to determine whether POET enables agents to solve problems of greater difficulty. We conclude that using POET facilitates learning better than direct optimization for the reinforcement learning environment Super Mario Bros when using NEAT for agent optimization.

6.2 Contributions

There are several contributions within this work. The most significant contribution is the study of minimal criteria and goal-switching in the Paired Open-Ended Trailblazer (POET) algorithm. An ablation study was performed to determine if the minimal criterion and transfer are necessary parts of POET. During the ablation study, the use of different minimal criteria and transfer frequencies were also explored to determine how they affect the coevolutionary process.

The work also contributes with the application of POET to a new problem domain. There have been limited previous applications of the algorithm; in this work, we use POET to successfully solve control problems for an obstacle course environment containing adversaries. Challenges solved and generated by our model for the new problem domain were found to pose a significant challenge even for advanced human players. We also contribute by using the Neuroevolution of Augmenting Topologies (NEAT) genetic algorithm for agent optimization in POET. This is the first time POET has been combined with an optimization method that searches for neural network topology and weights that we know of.

Previous work drawing inspiration from POET by Uber AI labs and Google DeepMind has utilized vast computing resources [29] [25]. In this work, we explored using POET with limited resources by studying how resources should be distributed. We also explored whether POET is applicable when employing a

limited amount of agent-environment pairs. The knowledge accumulated in this work may be used to guide future works where large-scale distributed processing is not available.

The final contribution is a POET framework implemented in Java. The framework will be available in a public git repository which can be found at https://github.com/jakobvaa.

6.3 Future Work

The most significant example of open-endedness that we know of is evolution. Organisms resulting from evolution use an indirect encoding for their genotype; DNA. Therefore it would seem valuable to explore the use of indirect encodings when trying to imitate an open-ended process. Direct encoding approaches like Neuroevolution of Augmenting Topologies employ a one-to-one mapping between parameter values of the neural network and its genetic representation. A disadvantage of one-to-one mappings is that parts of the solution that is similar must be discovered separately. Indirect encodings may allow the reuse of information, resulting in compact genetic representations. For future work, it could be interesting to use an indirectly encoded neuroevolution approach along with the Paired Open-Ended Trailblazer algorithm.

During the master's thesis, research and development were put towards integrating the indirectly encoded ES-HyperNEAT and HyperNEAT neuroevolution methods with POET in Java. The use of HyperNEAT would enable the agents to learn from high-dimensional input like raw-pixel data [10]. Most of the integration is finished, but the objective had to be abandoned because of time constraints and the scope of the master's thesis. The project integrating ES-HyperNEAT and HyperNEAT with POET can be granted by request.

Bibliography

[1] Uri Alon. Optimality and evolutionary tuning of the expression level of a protein Network topology View project. 2005.

[2] Jonathan C. Brant and Kenneth O. Stanley. Minimal criterion coevolution: A new approach to open-ended search. In *GECCO 2017 - Proceedings of the 2017 Genetic and Evolutionary Computation Conference*, pages 67–74. Association for Computing Machinery, Inc, 7 2017.

[3] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Evolving Competitive Car Controllers for Racing Games with Neuroevolution. 2009.

[4] Jeff Clune, Jean Baptiste Mouret, and Hod Lipson. The evolutionary origins of modularity. *Proceedings of the Royal Society B: Biological Sciences*, 280(1755), 3 2013.

[5] G Cybenkot. Mathematics of Control, Signals, and Systems Approximation by Superpositions of a Sigmoidal Function*. *Math. Control Signals Systems*, 2:303–314, 1989.

[6] Charles Darwin. On the Origin of Species, 1859. *On the Origin of Species, 1859*, 6 2004.

[7] David J Earl, Michael W Deem, and David Chandler. Evolvability is a selectable trait. Technical report, 2004.

[8] Faustino Gomez, J ¨ Urgen Schmidhuber, Juergen@idsia Ch, and Risto Miikkulainen. Accelerated Neural Evolution through Cooperatively Coevolved Synapses. *Journal of Machine Learning Research*, 9:937–965, 2008.

[9] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets.

[10] Matthew Hausknecht, Joel Lehman, Risto Miikkulainen, and Peter Stone. A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):355–366, 12 2014.

[11] Nadav Kashtan and Uri Alon. Spontaneous evolution of modularity and network motifs. *Proceedings of the National Academy of Sciences of the United States of America*, 102(39):13773–13778, 9 2005.

[12] Nadav Kashtan, Elad Noor, and Uri Alon. Varying environments can speed up evolution. *Proceedings of the National Academy of Sciences of the United States of America*, 104(34):13711–13716, 8 2007.

[13] Marc Kirschner and John Gerhart. The plausibility of life : resolving Darwin's dilemma. page 314, 2005.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks.

[15] Joel Lehman and Kenneth O Stanley. Exploiting Open-Endedness to Solve Problems Through the Search for Novelty. 2008.

[16] Joel Lehman and Kenneth O Stanley. Abandoning Objectives: Evolution Through the Search for Novelty Abandoning Objectives: Evolution Through the Search for Novelty Alone Alone Recommended Citation Recommended Citation Abandoning Objectives: Evolution Through the Search for Novelty Alone. 2011.

[17] Hod Lipson, Jordan B. Pollack, and Nam P. Suh. ON THE ORIGIN OF MODULAR VARIATION. *Evolution*, 56(8):1549–1556, 8 2002.

[18] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites.

[19] Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers Robotics AI*, 3(JUL), 7 2016.

[20] Christopher Rosin, Christopher D Rosin, and Richard K Belew. New methods for competitive coevolution. Technical report, 1997.

[21] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Van Den Driessche, Thore Graepel, and Hassabis Demis. Mastering the Game of Go without Human Knowledge.

[22] L. B. Soros and Kenneth O. Stanley. Identifying necessary conditions for open-ended evolution through the artificial life world of chromaria. In *Artificial Life 14 - Proceedings of the 14th International Conference on the Synthesis and Simulation of Living Systems, ALIFE 2014*, pages 793–800. MIT Press Journals, 2014.

[23] Russell K Standish. :23 WSPC/157-IJCIA alife-complexity-ijcia OPEN-ENDED ARTIFICIAL EVOLUTION. 2002.

[24] Kenneth O Stanley and Risto Miikkulainen. Evolving Neural Networks through Augmenting Topologies. Technical report, 2001.

[25] Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, Nat Mcaleese, Nathalie Bradley-Schmieg, Nathaniel Wong, Nicolas Porcel, Roberta Raileanu, Steph Hughes-Fitt, Valentin Dalibard, and Wojciech Marian Czarnecki. Open-Ended Learning Leads to Generally Capable Agents Open-Ended Learning Team*. Technical report.

[26] Togelius. The Mario AI Championship 2009-2012. 2013.

[27] Julian Togelius, Sergey Karakovskiy, and Robin Baumgarten. The 2009 Mario AI Competition. In *2010 IEEE World Congress on Computational Intelligence, WCCI 2010 - 2010 IEEE Congress on Evolutionary Computation, CEC 2010*, 2010.

[28] Julian Togelius, Sergey Karakovskiy, Jan Koutník, and Jürgen Schmidhuber. Super Mario evolution. In *CIG2009 - 2009 IEEE Symposium on Computational Intelligence and Games*, pages 156–161, 2009.

[29] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions. 2019.

Appendix

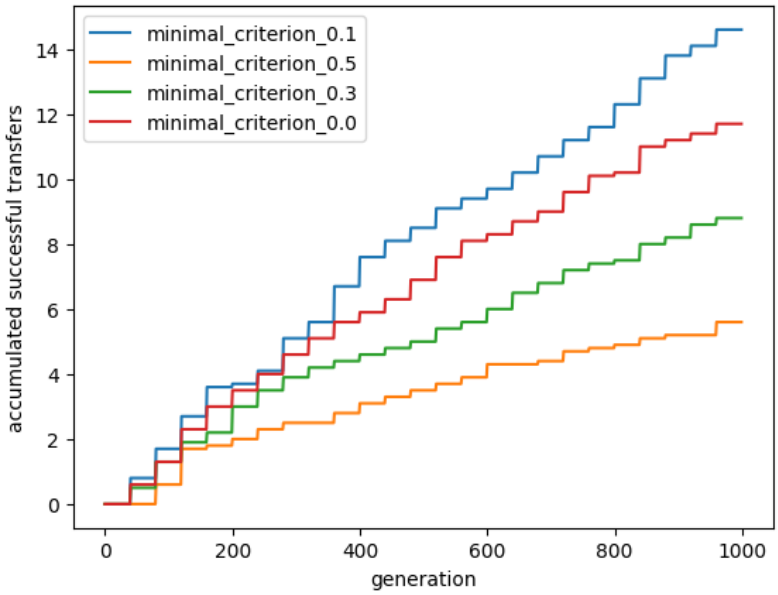

Figure 6.1: Experiment 3: Transfer success by generation. Transfers are more common when agents have shown little progress towards solving each environment.

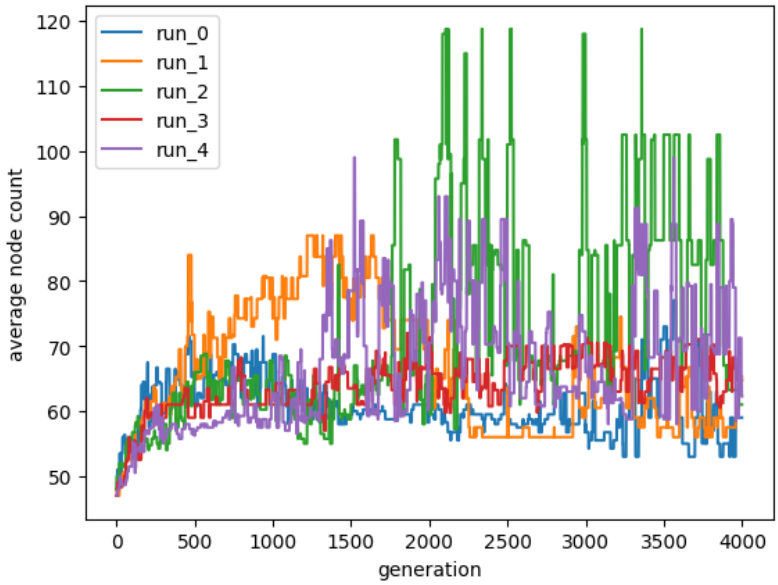

Figure 6.2: Experiment 4: Average amount of nodes in the phenotype for the top agents of each agent sub-population for each individual run.

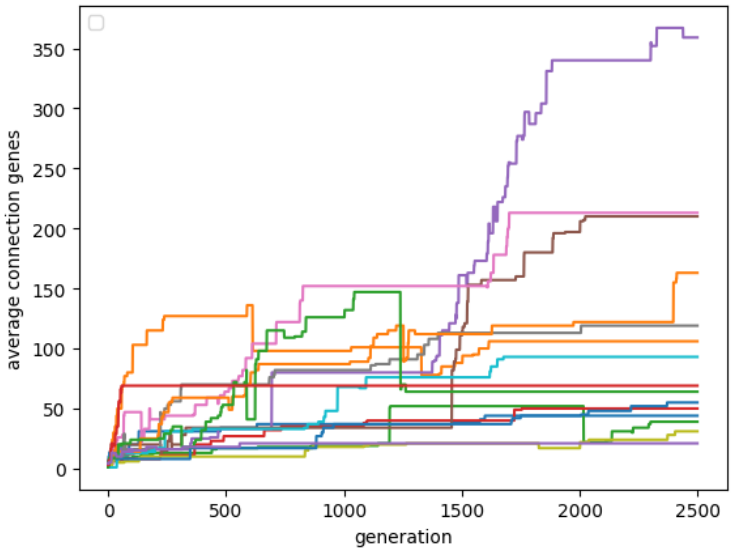

Figure 6.3: Experiment 6: Average amount of connection genes for each agent by generation. Each line represents one run of direct optimization by Neuroevolution of Augmenting Topologies for a unique hard environment.

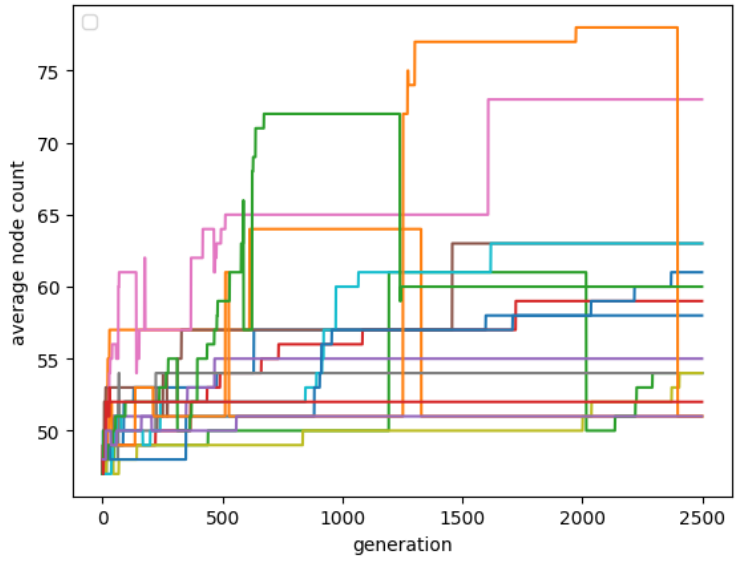

Figure 6.4: Experiment 6: Amount of nodes in the phenotype of the best performing agent for each level by generation. Each line represents one run of direct optimization by Neuroevolution of Augmenting Topologies for a unique hard environment.

Jakob Myklebust Vaardal

Exploring the Paired Open-Ended Trailblazer algorithm

NTNU
Norwegian University of
Science and Technology