FACULTY OF SCIENCES

Evolving Virtual Creatures through Geometrical Patterns

Dries Marzougui
Student number: 01606999

Supervisors: Prof. dr. Yvan Saeys, Prof. dr. ir. Francis wyffels
Counsellor: Dr. ing. Matthias Freiberger

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in de informatica

Academic year 2020-2021

GHENT UNIVERSITY

Abstract - Dutch

Een ambitieuze langetermijndoelstelling binnen het domein van kunstmatig leven (ALife) waarin met behulp van simulaties systemen worden onderzocht die gerelateerd zijn aan het natuurlijk leven, is het benaderen van evolutie zoals die in de biologische wereld wordt waargenomen. Alhoewel onderzoek in ALife alreeds veelbelovende resultaten heeft opgeleverd, waaronder het ontstaan van soorten en complexe interacties tussen virtuele organismen, heeft geen enkel werk een evolutionaire dynamiek bereikt die een intrinsieke aandrift naar toenemende diversiteit en complexiteit van virtuele organismen op lange termijn inhoudt. In deze masterproef zetten we een stap in de richting van het ontketenen van het potentieel van virtuele evolutie door het probleem van twee verschillende kanten te benaderen. Ten eerste pakken we de huidige problemen met betrekking tot de gelijktijdige evolutie van agent controle en morfologie aan. Hiervoor introduceren we een nieuwe methodologie die zowel de controller als de morfologie door eenzelfde genetische codering voorstelt: de *Compositional Pattern Producing Network* (CPPN). Gezien de keuze van de genetische codering, wordt de voorgestelde aanpak *One CPPN to Rule them All* (OCRA) genoemd. Deze methodologie dient niet alleen een doel in het domein van de evolutionaire robotica, maar maakt ook een geheel nieuwe waaier aan mogelijke interacties tussen wezens en ethologische gedragingen mogelijk. Dit als een gevolg van de fenotypische vrijheid die ze toelaat. Ten tweede introduceren we een uitgebreidere omgeving waarin de evolutie van deze wezens kan plaatsvinden. Deze omgeving is een modulair virtueel ecosysteem dat belangrijke stimuli bevat voor een toenemende diversiteit en complexiteit van de inwonende virtuele organismen. Vele van deze stimuli ontbreken in de systemen die vandaag de dag beschikbaar zijn. Hoewel de evolutionaire dynamiek zoals waargenomen in de biologische wereld nog ver buiten bereik ligt, kan de combinatie van de geïntroduceerde OCRA methodologie met de uitgebreidere omgeving dienen als een belangrijke opstap naar dit doel.

Abstract - English

An ambitious long-term goal for artificial life (ALife), which examines systems related to natural life and its processes through the use of simulations, is to approximate the potential of evolution as witnessed in the biological world. Yet while research in ALife has produced promising results, including the emergence of species and complex interactions between virtual organisms, no work has achieved an open-ended evolutionary dynamic involving a long-term, intrinsic drive for increased diversity and complexity of virtual organisms. In this thesis, we take a step towards unshackling the full potential of virtual evolution by approaching the problem from two different sides. First, we address the current issues regarding simultaneous evolution of agent control and morphology by introducing a powerful novel methodology that concomitantly represents both through a single genetic encoding: the *Compositional Pattern Producing Network* (CPPN). Granted the choice of the genetic encoding, the proposed approach is called *One CPPN to Rule them All* (OCRA). Besides serving purpose in the evolutionary robotics domain, this approach opens up a whole new range of possible creature interactions and ethological behaviors to occur due to the phenotypic freedom it allows for. Second, we introduce a more comprehensive environment in which the evolution of these creatures can take place. This environment is a modular virtual ecosystem that fundamentally incorporates important stimuli for increased diversity and complexity of its inhabiting virtual organisms, which are believed to be lacking in the available systems to date. Although the long-term evolutionary dynamics as observed in the biological world are still far beyond reach, the introduced OCRA methodology embedded in the more comprehensive environment can serve as an important stepping stone towards this goal.

Evolving Virtual Creatures through Geometrical Patterns

Dries Marzougui
Supervisors: Prof. dr. Yvan Saeys, Prof. dr. ir. Francis wyffels

***Abstract*—An ambitious long-term goal for artificial life (ALife), which examines systems related to natural life and its processes through the use of simulations, is to approximate the potential of evolution as witnessed in the biological world. Yet while research in ALife has produced promising results no work has achieved an open-ended evolutionary dynamic involving a long-term, intrinsic drive for increased diversity and complexity of virtual organisms. In this work, we approach the problem from two different sides. First, we introduce a novel methodology that addresses the current issues regarding complete creature evolution. This methodology is named *"One CPPN to Rule them All"* (OCRA). Second, we introduce a more comprehensive environment in which the evolution of these creatures can take place. This environment is a modular virtual ecosystem that inherently incorporates important stimuli for evolution that are believed to be missing in systems available to date.**

***Index Terms*—Artificial Life, Evolutionary Robotics**

I. Introduction

EVOLUTION sculpts both the body plans and nervous systems of agents together over time. By contrast in evolutionary robotics, the application of evolutionary optimization to include both the robot neural controllers and morphologies has proven to be problematic [1]. While recent successes have demonstrated the potential of effective optimization for the control policies of agents with fixed morphologies [2] or to a lesser extent the optimization of morphologies for agents with minimal and fixed control policies [3], the simultaneous optimization of the two has seen very limited success [4].

Cheney et al. [4] hypothesize that traditional evolutionary algorithms are hindered in this setting primarily due to an effect of embodied cognition, in which an individual's body plan and brain have an incentive to specialize their behaviors to complement one another. This specialization makes improvements to either subsystem difficult without complementary changes in the other (a highly unlikely event given current algorithms) and thus results in an embodied agent which is fragile with respect to design perturbations.

Following the hypotheses made by Cheney et al., we proposes an alternative approach to address the current issues of simultaneous evolution of agent brain and morphology. Current state-of-the-art approaches [1, 5, 6] separate the genetic encoding for morphology and controller as this enables more control over the optimization process (e.g. iteratively switching between morphology optimization and controller optimization). However, we hypothesize that evolving a single genetic encoding simultaneously representing both incorporates complementary changes between them in a more natural way, as both brain and body are then derived from the same higher-order parameter space. Consequently, this allows for a higher level of embodied intelligence as both brain and body are more acutely attuned to one another.

However, this requires a rather powerful genetic encoding. Based on an exhaustive study of related work within this domain, we put forth the *Compositional Pattern Producing Network* (CPPN) [7]. CPPNs have already successfully shown their ability to separately encode functional robot morphologies [3, 8] and controllers [9, 10] by representing them as *geometrical patterns* within some predefined space. Within this novel methodology, the CPPN is used to generate both morphology and controller defining patterns within the same geometric space. As the CPPN is then evolved to learn an appropriate relationship between the defining patterns of both the morphology and controller, we hypothesize that mutations on the CPPN should naturally reflect into complementary changes within both of these patterns. Given the choice of the genetic encoding, we named the methodology *"One CPPN to Rule them All"* (OCRA).

Besides serving purpose in the evolutionary robotics domain, a novel methodology that mitigates the current issues of complete agent evolution is beneficial for the artificial life (ALife) domain as well. One particular object of study within the ALife domain are a type of embodied agents, often referred to as *virtual creatures*. Within our work, virtual creatures are designated as the entity on which the proposed methodology is applied.

Furthermore, an environment is required in which these virtual creatures can be evolved. Within our biological world, evolution has shown its capability of leading to remarkable results. A (simplified) virtual twin of our world accordingly forms a straightforward choice for an environment in which evolutionary experiments can be studied. Such a virtual twin is often referred to as a virtual ecosystem. Within such an environment, the creatures are given the individual goal of surviving through collecting energy available in the environment. Collectively, their goal is to sustain their numbers through their own mating behaviors, thereby no longer requiring additional *artificial* creations to maintain a minimum amount of creatures. Although research concerning evolution in the domain of virtual worlds (virtual ecosystems being an important subset) has already led to some promising results, including the emergence of *species* and complex interactions between virtual organisms, it has not yet succeeded in reproducing the long-term evolutionary dynamics as observed in the biological

world.

One possible explanation is that the scale of these systems, both in terms of population sizes and duration of simulations, has simply not been large enough to date. However, T. Taylor [11] argues that the poor evolvability is not just due to issues of scale, but also to some more fundamental problems with the way in which these systems have been designed[1]. Specifically, he indicates the following three major issues:

1) The lack of complex dynamics within the environment.
2) The restrictiveness of ecological interactions, both between creatures and between creatures and their environment.
3) The lack of mixing genetic material between creatures.

In line with these considerations, we introduce a more comprehensive virtual ecosystem that fundamentally incorporates these important stimuli for increased diversity and complexity of its inhabiting virtual organisms.

II. Background

This section discusses relevant background concerning the CPPN and its usage as a genetic encoding within both the neuroevolution and morphological evolution domains.

A. The Compositional Pattern Producing Network (CPPN)

The CPPN, introduced by K. Stanley in [7], plays a central role within this work due to its usage as the single genetic encoding representing both agent controller and morphology. In essence, this is a typically small artificial neural network (ANN) in which each neuron can have a different activation function, such that the network as a whole makes a *composition* of different functions. This type of network can then be used to paint patterns within some n-dimensional space, by querying a value for each n-dimensional point within that space. Therefore, the CPPN receives the point's coordinates as input. The resulting patterns thereby reflect the same properties as exhibited by the underlying activation functions. Consequently, the choice of activation functions allows steering the patterns towards desirable properties such as symmetry, repetition, and repetition with variation [7]. These regularities make it an attractive indirect genetic encoding, both to represent morphologies and ANN-based controllers.

B. Neuroevolution

The usage of CPPNs as a genetic encoding was made possible by the *Neuroevolution of Augmenting Topologies* (NEAT) algorithm [12]. Being the first work to enable the application of the required evolutionary operators (mutation and crossover) to ANNs in a stable manner, NEAT can be seen as one of the fundamental genetic algorithms within the neuroevolution domain. Although the original intention of NEAT was to directly evolve the ANNs that represent the phenotypic candidate solutions, its usage of a direct encoding scheme severely disadvantaged it to grow larger ANNs. In direct encodings like NEAT, each part of the solution's representation (i.e. each *gene*) maps to a single piece of structure in the final solution [13].

This shortcoming of the NEAT approach was addressed by the *HyperNEAT* extension [13]. Instead of using NEAT to evolve the resulting ANNs directly, this extension uses NEAT to evolve an indirect encoding capable of representing the resulting ANNs. This indirect encoding was chosen to be the CPPN due to the desirable regularities that its patterns exhibit. The CPPN in HyperNEAT plays the role of DNA in nature, but at a much higher level of abstraction; in effect it encodes a pattern of weights that is painted across the geometry of a network [9]. In practice, the user of the algorithm has to define the placement of neurons within some geometrical space (often referred to as the *substrate*). The evolved CPPN is then used to query the weight of a connection between two neurons, by feeding it the coordinate locations of these two neurons. Formally, CPPNs can thus be seen as functions that are given the location of neurons (i.e. the geometry of a network) as input and return the weights between these locations as output. That way, when queried for many pairs of nodes organized within the substrate, the result is a topographic connectivity pattern within that substrate.

Despite its novel capabilities, a significant limitation with the original HyperNEAT approach is that the human user has to decide the placement and number of hidden neurons within the substrate. This issue was mitigated by an additional extension, called *evolvable-substrate HyperNEAT* (ES-HyperNEAT) [9], by showing that both connectivity *and* hidden node placement can automatically be determined by information already inherent in the pattern encoded by the CPPN. The basic philosophy is that *"density should follow information"*: where there is more information (variance) in the CPPN-encoded weight pattern, there should be a higher density of neurons and connections within the substrate to capture it [9]. In practice, the algorithm starts from a set of user-defined input and output neurons placed in the substrate. It then searches the pattern of connection weights for each such initial neuron, thereby connecting the neuron with existing or newly created hidden neurons on locations that lie within areas of high connection weight variance. Next, for a user-defined amount of iterations, it *discovers* new hidden neurons and connections starting from the previously discovered hidden neurons.

One remaining challenge to be addressed was to allow the evolved ANNs to learn during their usage, i.e. to change their connection weights. Building further upon the CPPNs encoding power, the *adaptive ES-HyperNEAT* (AESHN) extension proposed in [9], augments the CPPN outputs in such a way that it also produces patterns of connection-specific Hebbian learning rules (next to the original patterns of connection weights).

C. Morphological Evolution

As described in Section II-A, CPPNs have the rather unique ability of being able to paint patterns within some geometrical

[1] Although many other elements play a role in the elusiveness of obtaining such an open-ended evolutionary process, the focus of study here are the problems inherent to the system or environment in which evolution takes place.

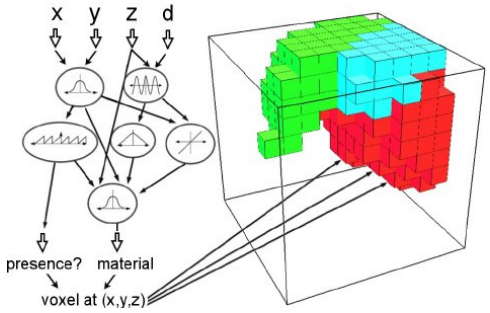


Fig. 1: Visual example of how a CPPN encoding is mapped to a 3D (soft robot) morphology, as given in [3]. The CPPN is iteratively queried for each voxel within a bounding area and produces output values as a function of the coordinates of that voxel. These outputs determine the presence of voxels and their material properties to specify a soft robot morphology.

space. Based on the activation functions we supply them with, the produced patterns exhibit properties such as symmetry, asymmetry and repetition (with variation) of segments. As these are attractive properties for robot morphologies, the CPPN establishes a promising genetic encoding for evolving robot morphologies and has successfully been used within both the rigid [5, 14] and soft robotics fields [3, 8]. Figure 1 further concretizes by illustrating how the CPPN encoding in [3] is mapped to the morphology.

III. One CPPN to Rule them All (OCRA)

The proposed OCRA methodology mitigates the issues of simultaneous brain-body evolution hypothesized by Cheney et al. [4] by representing both the ANN-based controller and morphology with a single genetic encoding: the CPPN. Therefore, we combine the CPPN based AESHN [9] neuroevolution algorithm, with the CPPN based morphological evolution approach of [3]. This is done by merging their proposed CPPN architectures into one, thereby allowing a single CPPN to produce both controller and morphology defining patterns. The main difference in terms of morphology is the types of cube-shaped building blocks we make available, which in our case results in rigid morphologies composed out of connected and undeformable building blocks. While many possible types of building blocks can be designed by both humans and evolved agents themselves, the major types used here are:

- **Filler Block**: a common block used to "fill" space, without any specific function other than to connect to other blocks.
- **Brain Block**: a unique block that defines the substrate. This is subdivided into an inner and outer region. Whereas the inner region is reserved for hidden neurons evolved by the AESHN algorithm, the outer region can be used to place user-defined input and output neurons.
- **Joint Block**: a block that contains a movable joint. The parameters describing the joint's degrees of freedom are extracted from CPPN induced patterns. This block hosts both input and output neurons. Whereas the input neurons are given the current orientation and position of the block relative to the brain block[2], the output neurons control the movable joint.

While the original idea was to include a *sensor block* that allows the creature to evolve perception of its environment itself, we reduced the initial complexity during experimentation (cfr. Section V) by fixing the sensors to the brain block and providing the corresponding observations to the controller through user-defined input neurons.

To extract a complete creature from its CPPN encoding, we first build the morphology by querying the CPPN for each voxel location within a box-shaped space (centered around origin), similar to the process illustrated in Figure 1. This in turn gives us all information required in order to build the ANN-based controller using the AESHN algorithm: a substrate (the brain block) and the locations of both the morphology-dependent and user-defined input and output neurons.

IV. Towards a more comprehensive Virtual Ecosystem

Next to a novel approach that better allows evolving creatures as a whole, we introduce a more comprehensive environment in which that evolutionary process can take place. This environment is a virtual ecosystem made in Unity [16], that fundamentally incorporates the missing components of the virtual worlds in which evolution has been studied to date (cfr. Section I). An example instantiation of the virtual ecosystem is shown in Figure 2.

In the design of the environment, we take a bottom-up approach that defines the whole virtual ecosystem (and its inhabitants) by means of modular building blocks. The key idea being that such a modular world allows to more easily introduce complex environmental dynamics, such as ever-changing landscapes due to blocks being moved around or stochastically changing the properties of the inserted blocks. The building blocks used here include *earth*, *water*, *wall* and *randomly defined* blocks. Patches of earth blocks establish the land on which the artificial organisms live, while patches of water blocks form the "oceans" in between those patches of land. Wall blocks can be used to create a visible boundary of the environment, next to inserting impassable barriers to isolate parts of the inhabiting population. Such impassable barriers are known to stimulate the emergence of species [17]. The purpose of the randomly defined blocks is discussed below.

A. (Inter)actions

The restrictiveness of interactions in many works originates from the usage of a predefined or "hard-coded" set of actions for the creatures. In other words, a limited range of actions limits the range of interactions. The need for such a predefined set of actions was alleviated by relying on the phenotypic freedom of creatures that methods such as the proposed OCRA approach grant the evolutionary process. This phenotypic freedom induced by the simultaneous evolution of both creature morphology and controller introduces a theoretically unlimited range of interactions between creatures. The key insight behind this being that the evolution of creatures can now lead to creatures discovering these actions and interactions themselves.

[2]This is inspired by a natural effect called *kinesthesia* [15].

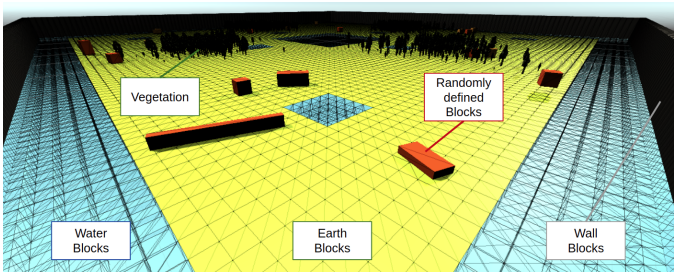


Fig. 2: A rendered (wireframe) view of an example virtual ecosystem. This is a rather small and simple environment, given that it only consists of 102×102 blocks and has a flat surface. All available block types are shown and annotated.

Furthermore, combining this phenotypic freedom with a modular virtual ecosystem composed out of mutable building blocks naturally opens up a whole new range of interactions between the creature and its environment. One type of environment block, the *randomly defined block*, was specifically designated for this purpose. The randomness relates to its geometry, i.e. its size and shape. Upon initialization of the environment, a user-defined amount of such randomly generated blocks is inserted. These can then be used by the creatures for whatever they see fit, such as potentially constructing "barriers" to hold off predators.

Nevertheless, some basic actions are initially predefined to lower the complexity threshold of creatures surviving in the virtual ecosystem. These included *eating*, *drinking*, *attacking* and *mating*. While eating and drinking had trivial meanings, the attacking and mating actions allow to naturally incorporate two important (inter-creature) interaction-related concepts: *predation* (creatures can kill and eat each other) and *sexual reproduction*. Sexual reproduction forms the main strategy for letting creatures reach their collective goal (maintaining a minimum population size), next to allowing a "vertical" mixture of genetic material (offspring carries on mutated and crossed over genes of its parents).

B. Energy

Within this virtual ecosystem, creatures have three types of energy: (1) *health-energy*, (2) *food-energy* and (3) *drink-energy*. The maximum amount of energy that a creature can store is equal for all of these types and scales with the amount of building blocks that its morphology is composed of (morphological complexity). The state of all three of these energy types are provided as additional inputs to the controller of the creature, allowing it to change its behavior depending on its energy state.

In order to survive, a virtual creature must keep its health-energy from reaching zero, which can happen due to "starvation", "dehydration" or being attacked by other creatures. Whereas starvation occurs when a creature's food-energy has reached zero, dehydration occurs when a creature's drink-energy has reached zero. Both of these independently reduce the health-energy at a constant rate, such that when both occur simultaneously, these rates are coalesced. Health-energy is regenerated when food- and drink-energy levels are both above zero; thereby draining these two types of energy. A creature dies when it has reached its maximum lifespan (scales with morphological complexity) or if its health-energy reaches zero. This however allows the possibility of dying while still maintaining relatively high food- and drink-energy values (for instance if health-energy was driven to zero due to an attack). A creature's remaining food and drink-energy remains stored in its carcass, such that other creatures (e.g. the attacker) can consume it. This mechanism makes the predator-prey interactions quite natural. Whereas food-energy can be replenished by *eating* vegetation (discussed below), drink-energy is replenished by drinking from *water* blocks. Both types of energy can be gained simultaneously by *eating* a carcass.

Both food- and drink-energy are depleted at a constant "metabolism" rate, as creatures could otherwise live out their full lifespan without taking actions to survive. However, food-energy is additionally depleted by taking actions. The amount of energy expended by *reproduction*, *attacking* and morphological actions, i.e. joint movements, scales with the complexity of both the morphology (amount of building blocks) and the controller (amount of neurons and connections). While such actions can consequently cause starvation to occur more quickly, if creatures take appropriate actions (e.g. moving towards food and drinks) they can increase their lifespan by responding to the pressure of the constant metabolism rate.

C. Vegetation

Besides the virtual creatures, the presented virtual ecosystem hosts another type of virtual organism that is subject to evolution, although at a much lower level of complexity. Besides the carcasses of other creatures, this organism serves as the main food source of the virtual creatures and is portrayed as stochastically spreading vegetation. The main interaction between these virtual organisms being that creatures can eat vegetation in order to replenish their food-energy. Moreover, an additional set of interactions originates from the ability of creatures to move vegetation around. These interactions, combined with the genetic variation induced by the evolution of both of these virtual organisms, serves as an additional stimulus for the evolution of both of these organisms.

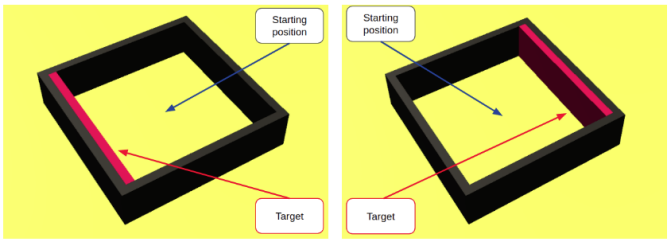
V. Experiments

This section discusses one of the most important experiments carried out on our path towards validating the proposed OCRA approach to evolve complete creatures. Many other experiments were carried out as well, but are excluded for brevity.

A. Evolving Perception

Given the positive validation of the OCRA approach to completely evolve creatures capable of movement[3] and moving towards given target locations in an earlier series of experiments, the next series of experiments was concerned with evaluating its ability to evolve creatures that appropriately perceive their

[3]A video demonstrating the movement of one of the evolved creatures is available at: https://www.shorturl.at/dmoCJ



(a) Trial 1 (b) Trial 2

Fig. 3: The Red-Wall environment, made in Unity [16]. Within this environment, each creature is evaluated during two consecutive trials in which it must reach the target wall (red). As shown, the location of the target wall is different within each of these two trials. Consequently, to solve this environment the creatures must evolve towards appropriately using their sensor observations.

environment. To do so, we created a simple environment in which the creature must use its sensor observations to locate and move to a target during two consecutive trials. This environment was named "Red-Wall" and is further illustrated in Figure 3. The *fitness* of a creature in each trial is set to 1 if it reaches the target, with an additional factor that increases the fitness based on the number of remaining time steps (thereby encouraging faster creatures). If the creature was not able to reach the target within a trial, its fitness for that trial is calculated based on how much closer it got to the target compared to its starting location (0 being equal to not having gotten closer and 0.99 to nearly touching it). The total fitness of a creature is then set as the minimum of its fitness scores of its two trials, thereby preventing insignificant or "lucky" creatures to receive a high fitness.

While the original intention was to allow for *sensor blocks* to be evolved within the creature morphology, i.e. letting the OCRA approach evolve the placement of sensors as well, we reduce complexity by fixing the sensors to the brain block. The corresponding observations are fed to the controller through user-defined input neurons. The sensors used here are *raycast* sensors, i.e. sensors that cast rays into the environment that return the type of object they collide with.

Experimental Setup

CVT-MAP-Elites [18] was used as the underlying framework to guide the evolutionary process. The employed behavior descriptor consisted out of (1) a number of direction samples, i.e. samples that indicate towards which direction the creature was moving at certain points in time, representing creature trajectory and (2) the (normalized) number of filler and joint blocks within each quadrant of the creature's morphology. The elites archive size was set to 100. Population size was set to 128. To increase the soundness of the results, we conducted eight runs of this experiment. Each such run was done on a compute node with four *Intel Xeon E5-2650 v2 @ 2.60 GHz* CPUs, using multiprocessing to take maximal advantage of the 32 cores available.

Results

Three out of eight runs we conducted were able to evolve creatures capable of reaching the target in both of their trials. While it is not surprising that not all runs lead to equally successful results, their failure was shown to originate from a lack of genotypic (CPPN) complexity. Given that all mutations occur stochastically, this issue is a natural artifact of the evolutionary process.

A schematic overview of the best performing creature[4] over all runs is shown in Figure 4. This creature was evolved after 392 generations (equivalent to 50176 evaluations given the population size of 128) and obtained a fitness score of 1.634. This fitness means that it completed both its trials successfully, with its "worst" trial only requiring 36% of the given time steps. The run that evolved this creature also evolved 187 other creatures that solved the environment, spanning over 15 different morphological configurations. Next to this run, a different run evolved creatures with similar morphologies and controller complexities, but different movement behaviors[5]. The final successful run differentiates itself from the other runs, by evolving towards much higher controller complexities of up to approximately 400 hidden neurons and 170000 connections. Although the controllers were much more complex, the morphologies and obtained fitness scores remained similar to other runs. An example creature of this run is shown in Figure 5. Interesting to note is that the controllers evolved within this run show a modular structure. Intuitively, this modular structure is composed out of a memory module (recurrent neurons), a learning module (neuromodulatory neurons) and a processing module (standard neurons). While this structure is a rather interesting product of the evolutionary process, next to being visually appealing and somewhat resembling the modular structure of biological brains, the environment in which it was evolved is considered to be too simple to truly investigate its potential benefits. However, given that this creature was already evolved after 42 generations, only a maximum of 42 mutations to its underlying CPPN genome could have occurred. Additionally, the CPPN of the creature shown here did not have any hidden neurons. This suggests that its performance mainly originates from the powerful encoding capabilities of the underlying CPPN genome from which this modular structure is derived.

VI. Discussion

Whereas the proposed OCRA approach underwent a thorough series of experiments, experiments within the proposed virtual ecosystem always took place in heavily simplified versions of it. The main reason for the simplification being the novelty of the OCRA approach. Given its novelty, we cannot expect it to immediately take on the complexity of the complete virtual ecosystem and lead to successful results. Nevertheless, the complete virtual ecosystem that was presented here is fully implemented and stands ready for further experimentation[6].

[4]A video demonstrating its evaluation is available at: https://www.shorturl.at/dvR01

[5]A video demonstrating this rather peculiar, but successful movement is available at: https://www.shorturl.at/yHMR4

[6]The virtual ecosystem, and all other code developed throughout this thesis, is available at: https://github.ugent.be/dmarzoug/thesis

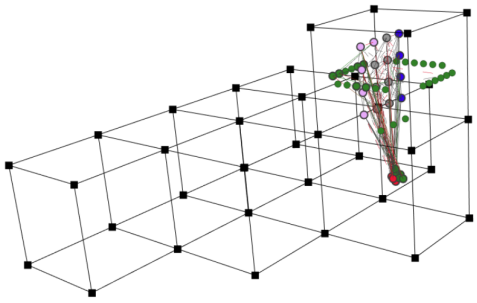


Fig. 4: Schematic visualization of the best performing creature of the "Evolving Perception" experiment. This creature's morphology is composed out of 7 filler blocks and 1 joint block underneath the brain block. The controller evolved within the brain block has 7 standard neurons (grey), 4 recurrent neurons (blue), 5 neuromodulatory neurons (pink) and 171 connections in total. Neuromodulatory neurons change the connection weights of other (non-neuromodulatory) neurons depending on their own activation, thereby allowing one brain region (e.g. where sensory information is processed) to directly influence the learning of another region (e.g. where movement is determined).

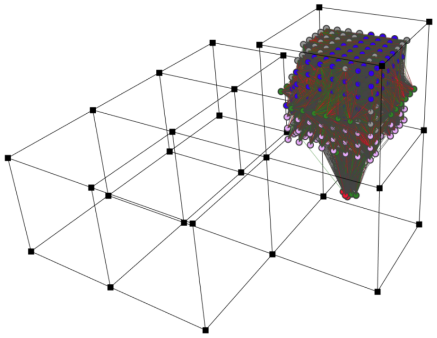


Fig. 5: Schematic visualization of an alternative creature evolved within the "Evolving perception" experiment. This creature's controller contains 43 standard neurons (grey), 357 recurrent neurons (blue), 143 neuromodulatory neurons (pink) and 167409 connections in total.

Given its biologically inspired setup, inherent scalability and extensibility, the presented environment can serve as a test bed for many future evolutionary works within the artificial life domain. It is our hope that some of these works will include the proposed OCRA approach, as we believe that this environment can unlock its true potential.

VII. Conclusion

Although we are aware that the long-term evolutionary dynamics as observed in the biological world are still far beyond reach, we believe that the combination of the proposed OCRA methodology and the more comprehensive virtual ecosystem will allow us to gain important new insights that may help to illuminate the path towards this ambitious long-term goal.

References

[1] Nick Cheney, Josh Bongard, Vytas SunSpiral, and Hod Lipson. Scalable co-optimization of morphology and control in embodied machines. *J. R. Soc. Interface*, 15(143), June 2018.

[2] Antoine Cully, Jeff Clune, and Jean-Baptiste Mouret. Robots that can adapt like natural animals. *CoRR*, abs/1407.3501, 2014.

[3] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. *SIGEVOlution*, 7(1):11–23, August 2014.

[4] Nicholas Cheney, Josh Bongard, Vytas Sunspiral, and Hod Lipson. On the difficulty of Co-Optimizing morphology and control in evolved virtual creatures, 2016.

[5] B Weel, E Crosato, J Heinerman, E Haasdijk, and A E Eiben. A robotic ecosystem with evolvable minds and bodies. In *2014 IEEE International Conference on Evolvable Systems*, pages 165–172, December 2014.

[6] Allan Zhao, Jie Xu, Mina Konaković-Luković, Josephine Hughes, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. RoboGrammar: graph grammar for terrain-optimized robot design. *ACM Trans. Graph.*, 39(6):1–16, November 2020.

[7] Kenneth O Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genet. Program. Evolvable Mach.*, 8(2):131–162, June 2007.

[8] Sam Kriegman, Douglas Blackiston, Michael Levin, and Josh Bongard. A scalable pipeline for designing reconfigurable organisms. *Proc. Natl. Acad. Sci. U. S. A.*, 117(4):1853–1859, January 2020.

[9] S. Risi and K. Stanley. An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. *Artificial Life*, 18:331–363, 2012.

[10] Pablo Reyes and María-José Escobar. Neuroevolutive algorithms for learning gaits in legged robots. *IEEE Access*, 7:142406–142420, 2019.

[11] Tim Taylor. Evolution in virtual worlds. *CoRR*, abs/1710.06055, 2017.

[12] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, 2002.

[13] Kenneth Stanley, David D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15:185–212, 02 2009.

[14] Daniel Richards and Martyn Amos. Evolving morphologies with CPPN-NEAT and a dynamic substrate. *Artificial Life Conference Proceedings*, 26:255–262, July 2014.

[15] Jia Han, Gordon Waddington, Roger Adams, Judith Anson, and Yu Liu. Assessing proprioception: A critical review of methods. *Journal of Sport and Health Science*, 5(1):80–90, 2016.

[16] Unity Technologies. Unity.

[17] Larry Yaeger. Computational genetics, physiology, metabolism, neural systems, learning, vision, and behavior or polyworld: Life in a new context. 03 1995.

[18] Vassilis Vassiliades, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. Using centroidal voronoi tessellations to scale up the multidimensional archive of phenotypic elites algorithm. *IEEE Transactions on Evolutionary Computation*, 22:623–630, 2018.

Evolutie van Virtuele Wezens via Geometrische Patronen

Dries Marzougui
Supervisors: Prof. dr. Yvan Saeys, Prof. dr. ir. Francis wyffels

***Samenvatting*—Een ambitieus langetermijndoel voor kunstmatige leven (ALife), het domein dat systemen onderzoekt die gerelateerd zijn aan het natuurlijk leven door middel van simulaties, is het benaderen van het potentieel van evolutie zoals waargenomen in de biologische wereld. Alhoewel onderzoek in ALife alreeds veelbelovende resultaten heeft opgeleverd, heeft geen geen enkel werk een evolutionaire dynamiek met een intrinsieke drang naar een grotere diversiteit en complexiteit van virtuele organismen op de lange termijn bereikt. In dit werk benaderen we het probleem van twee verschillende kanten. Eerst introduceren we een nieuwe methodologie die de huidige problemen met betrekking tot de volledige evolutie van virtuele wezens aanpakt. Deze methodologie heeft *"One CPPN to Rule them All"* (OCRA). Ten tweede introduceren we een uitgebreidere omgeving waarin de evolutie van deze wezens kan plaatsvinden. Deze omgeving is een modulair virtueel ecosysteem dat inherent belangrijke stimuli voor evolutie bevat, waarvan wordt gesteld dat ze ontbreken in de huidige systemen.**

***Index*—Kunstmatig Leven, Evolutionaire Robotica**

I. Introductie

EVOLUTIE ontwerpt in de loop van de tijd gelijktijdig het lichaam als het zenuwstelsel van agents. Daarentegen is in evolutionaire robotica de toepassing van evolutionaire optimalisatie om zowel de neurale robotcontrollers als morfologieën te omvatten problematisch gebleken [1]. Hoewel recente successen het potentieel hebben aangetoond van effectieve optimalisatie van het controlebeleid van agents met vaste morfologieën [2] of, in mindere mate, de optimalisatie van morfologieën voor agents met een minimaal en vast controlebeleid [3], heeft de gelijktijdige optimalisatie van de twee slechts zeer beperkt succes gekend [4].

Cheney et al. [4] veronderstellen dat traditionele evolutionaire algoritmen in dit scenario voornamelijk worden gehinderd door een effect van belichaamde cognitie, waarbij het lichaam en het brein van een individu een stimulans hebben om hun gedrag zo te specialiseren dat ze elkaar beter aanvullen. Deze specialisatie maakt verbeteringen aan een van beide subsystemen moeilijk zonder complementaire veranderingen in de andere (een hoogst onwaarschijnlijke gebeurtenis gezien de huidige algoritmen) en resulteert dus in een belichaamde agent die kwetsbaar is met betrekking tot het ontwerp.

In navolging van de hypothesen van Cheney et al., stellen we een alternatieve benadering voor om de huidige problemen van gelijktijdige evolutie van het brein en de morfologie van agents aan te pakken. De huidige state-of-the-art methodes [1, 5, 6] scheiden de genetische codering voor morfologie en controller, omdat dit meer controle over het optimalisatieproces mogelijk maakt (bijvoorbeeld iteratief schakelen tussen morfologie-optimalisatie en controller-optimalisatie). Wij veronderstellen echter dat het ontwikkelen van een enkele genetische codering die tegelijkertijd beide vertegenwoordigt, complementaire veranderingen daartussen op een natuurlijkere manier incorporeert, aangezien zowel de controller als de morfologie dan worden afgeleid van dezelfde parameterruimte van hogere orde. Bijgevolg zorgt dit voor een hoger niveau van belichaamde intelligentie, omdat zowel de controller als de morfologie scherper op elkaar zijn afgestemd.

Dit vereist echter een vrij krachtige genetische codering. Op basis van een uitgebreide studie van gerelateerd werk binnen dit domein, kiezen we hiervoor het *"Compositional Pattern Producing Network"* (CPPN) [7]. CPPNs hebben al met succes aangetoond dat ze functionele robotmorfologieën [3, 8] en -controllers [9, 10] afzonderlijk kunnen coderen door ze te definiëren als *geometrische patronen* binnen een vooraf gedefinieerde ruimte. Binnen deze nieuwe methodologie wordt de CPPN gebruikt om zowel morfologie- als controllerdefiniërende patronen binnen dezelfde geometrische ruimte te genereren. Aangezien de CPPN vervolgens wordt geëvolueerd om een geschikte relatie te leren tussen de bepalende patronen van zowel de morfologie als de controller, veronderstellen we dat mutaties op de CPPN van nature zouden moeten reflecteren in complementaire veranderingen binnen beide patronen. Gezien de keuze voor de genetische codering hebben we de methodiek *"One CPPN to Rule them All"* (OCRA) genoemd.

Naast het dienen van een doel in het domein van de evolutionaire robotica, is een nieuwe methodologie die de huidige problemen van de volledige evolutie van agents verlicht ook gunstig voor het domein van het kunstmatige leven (ALife). Een specifiek object van studie binnen het ALife-domein zijn een soort belichaamde agents, vaak *virtuele wezens* genoemd. Binnen ons werk zijn deze virtuele wezens aangewezen als de entiteit waarop de voorgestelde methodiek wordt toegepast.

Bovendien is een omgeving vereist waarin deze virtuele wezens worden geëvolueerd. Binnen onze biologische wereld heeft evolutie laten zien dat het tot opmerkelijke resultaten kan leiden. Een (vereenvoudigde) virtuele tweeling van onze wereld vormt dan ook een voor de hand liggende keuze voor een omgeving waarin evolutionaire experimenten kunnen worden bestudeerd. Zo'n virtuele tweeling wordt vaak een virtueel ecosysteem genoemd. Binnen een dergelijke omgeving krijgen de wezens het individuele doel om te overleven door energie te verzamelen die beschikbaar is in de omgeving. Gezamenlijk is hun doel om hun aantal in stand te houden door hun eigen paargedrag, waardoor er geen extra kunstmatige

creaties meer nodig zijn om een minimum aantal wezens in stand te houden. Alhoewel onderzoek naar evolutie in het domein van virtuele werelden (virtuele ecosystemen zijn een belangrijke subset) al tot veelbelovende resultaten heeft geleid, waaronder de opkomst van *soorten* van en complexe interacties tussen virtuele organismen, is het er nog niet in geslaagd om de evolutionaire dynamiek op lange termijn te reproduceren zoals waargenomen in onze biologische wereld.

Een mogelijke verklaring is dat de schaal van deze systemen, zowel wat betreft populatiegrootte als duur van simulaties, tot nu toe simpelweg niet groot genoeg is. T. Taylor [11] stelt echter dat de slechte evolueerbaarheid niet alleen te wijten is aan schaalbaarheidsproblemen, maar ook aan enkele meer fundamentele problemen met de manier waarop deze systemen zijn ontworpen[1]. Concreet geeft hij de volgende drie grote problemen aan:

1) Het ontbreken van complexe dynamiek binnen de omgeving.
2) De restrictiviteit van ecologische interacties, zowel tussen wezens als tussen wezens en hun omgeving.
3) Het gebrek aan vermenging van genetisch materiaal tussen wezens.

In overeenstemming met deze overwegingen introduceren we een uitgebreider virtueel ecosysteem dat deze belangrijke stimuli voor een grotere diversiteit en complexiteit van de inwonende virtuele organismen fundamenteel opneemt.

II. Achtergrond

Deze sectie bespreekt de relevante achtergrond met betrekking tot de CPPN en het gebruik ervan als genetische codering binnen zowel de neuro-evolutie- als de morfologische evolutiedomeinen.

A. Het Compositional Pattern Producing Network (CPPN)

De CPPN, geïntroduceerd door K. Stanley in [7], speelt een centrale rol in dit werk vanwege het gebruik ervan als de genetische codering die zowel de agent's controller als de morfologie vertegenwoordigt. In wezen is dit een typisch klein artificieel neuraal netwerk (ANN) waarin elk neuron een andere activatiefunctie kan hebben, zodat het netwerk als geheel een samenstelling maakt van verschillende functies. Dit type netwerk kan vervolgens worden gebruikt om patronen te schilderen binnen een n-dimensionale ruimte, door een waarde op te vragen voor elk n-dimensionaal punt in die ruimte. Daarvoor ontvangt de CPPN de coördinaten van het punt als invoer. De resulterende patronen weerspiegelen daarbij dezelfde eigenschappen als vertoond door de onderliggende activatiefuncties. Bijgevolg maakt de keuze van activatiefuncties het mogelijk om de patronen te sturen naar gewenste eigenschappen zoals symmetrie, herhaling en herhaling met variatie [7]. Deze regelmatigheden maken het een aantrekkelijke (indirecte) genetische codering, zowel om morfologieën als ANN gebaseerde controllers te encoderen.

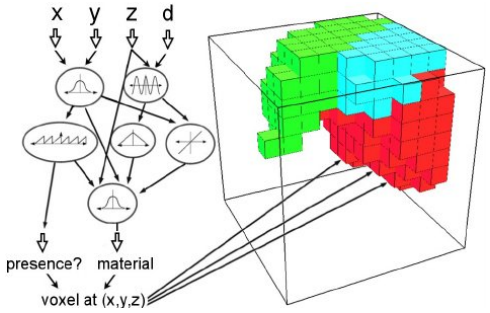
[1]Hoewel veel andere elementen een rol spelen bij de ongrijpbaarheid van het verkrijgen van een open evolutionair proces, ligt de focus van de studie hier op de problemen die inherent zijn aan het systeem of de omgeving waarin evolutie plaatsvindt.

B. Neuro-evolutie

Het gebruik van CPPN's als genetische codering werd mogelijk gemaakt door het *Neuroevolution of Augmenting Topologies* (NEAT) algoritme [12]. Als eerste werk dat de toepassing van de vereiste evolutionaire operatoren (mutatie en crossover) op ANN's op een stabiele manier mogelijk maakt, kan NEAT worden gezien als een van de fundamentele genetische algoritmen binnen het neuro-evolutiedomein. Hoewel de oorspronkelijke bedoeling van NEAT was om de ANN's die de fenotypische kandidaat-oplossingen vertegenwoordigen rechtstreeks te ontwikkelen, heeft het gebruik van een direct coderingsschema het ernstig benadeeld om grotere ANN's te laten creëren. In directe coderingen zoals NEAT, wordt elk deel van de representatie van de oplossing (d.w.z. elk *gen*) toegewezen aan één enkel stuk structuur in de uiteindelijke oplossing [13].

Deze tekortkoming van de NEAT-aanpak werd verholpen door de *HyperNEAT*-extensie [13]. In plaats van NEAT te gebruiken om de resulterende ANNs rechtstreeks te evolueren, gebruikt deze extensie NEAT om een indirecte codering te evolueren die de resulterende ANNs representeert. Deze indirecte codering is de CPPN vanwege de wenselijke regelmatigheden die de patronen vertonen. De CPPN in HyperNEAT speelt de rol van DNA in de natuur, maar op een veel hoger abstractieniveau; in feite codeert het een patroon van gewichten dat over de geometrie van een netwerk is geschilderd [9]. In de praktijk moet de gebruiker van het algoritme de plaatsing van neuronen binnen een geometrische ruimte (vaak het *substraat* genoemd) definiëren. De geëvolueerde CPPN wordt vervolgens gebruikt om het gewicht van een verbinding tussen twee neuronen te bepalen, door het de coördinaatlocaties van deze twee neuronen te voeden. Formeel kunnen CPPNs dus worden gezien als functies die de locatie van neuronen (de geometrie van een netwerk) als invoer krijgen en de gewichten tussen deze locaties als uitvoer teruggeven.

Ondanks zijn nieuwe mogelijkheden, is een belangrijke beperking met de originele HyperNEAT-aanpak dat de menselijke gebruiker de plaatsing en het aantal verborgen neuronen in het substraat moet beslissen. Dit probleem werd verholpen door een extra extensie, genaamd *evolvable-substrate HyperNEAT* (ES-HyperNEAT) [9], door aan te tonen dat zowel de connectiviteit als de plaatsing van verborgen neuronen automatisch kunnen worden bepaald door informatie die al inherent is aan het patroon geëncodeerd door de CPPN. De basisfilosofie is dat "dichtheid moet informatie volgen": waar er meer informatie (variantie) is in het CPPN-gecodeerde gewichtspatroon, zou er een hogere dichtheid van neuronen en verbindingen in het substraat moeten zijn om het vast te leggen [9]. In de praktijk gaat het algoritme uit van een verzameling door de gebruiker gedefinieerde input- en outputneuronen die in het substraat zijn geplaatst. Vervolgens doorzoekt het het patroon van verbindingsgewichten voor elk van deze initiële neuronen, waardoor de huidige neuron wordt verbonden met bestaande of nieuw gecreëerde verborgen neuronen op locaties in gebieden met een grote variantie in verbindingsgewichten. Vervolgens ontdekt het voor een (door de gebruiker gedefinieerd) aantal iteraties nieuwe verborgen neuronen en verbindingen vanaf de



Figuur 1: Visueel voorbeeld van hoe een CPPN-codering wordt omgezet in een 3D-morfologie (zachte robot), zoals getoond in [3]. De CPPN wordt iteratief opgevraagd voor elke voxel binnen een begrenzingsgebied en produceert uitvoerwaarden als een functie van de coördinaten van die voxel. Deze outputs bepalen de aanwezigheid van voxels en hun materiaaleigenschappen om een zachte robotmorfologie te specificeren.

eerder ontdekte verborgen neuronen.

Een resterende uitdaging die moest worden aangepakt, was om de geëvolueerde ANNs te laten leren tijdens hun gebruik. De *adaptive ES-HyperNEAT* (AESHN) extensie voorgesteld in [9], doet dit door de CPPN extra waarden te laten genereren op een zodanige manier dat het ook patronen van verbindingsspecifieke Hebbian leerregels produceert (naast de originele patronen van verbindingsgewichten).

C. Morfologische Evolutie

Zoals beschreven in Paragraaf II-A, hebben CPPNs het nogal unieke vermogen om patronen te schilderen binnen een geometrische ruimte. Op basis van de activatiefuncties die we ze toelaten, vertonen de geproduceerde patronen eigenschappen zoals symmetrie, asymmetrie en herhaling (met variatie) van segmenten. Omdat dit aantrekkelijke eigenschappen zijn voor robotmorfologieën, is de CPPN een veelbelovende genetische codering voor evoluerende robotmorfologieën en is het met succes gebruikt binnen zowel het stijve [5, 14] als het zachte [3, 8] roboticaveld. Figuur 1 concretiseert verder door te illustreren hoe de CPPN-codering in [3] wordt omgezet in een morfologie.

III. One CPPN to Rule them All (OCRA)

De voorgestelde OCRA-methodologie verzacht de problemen van gelijktijdige evolutie van agent controller en morfologie, door zowel de op ANN gebaseerde controller als de morfologie te definiëren met één enkele genetische codering: de CPPN. Daarvoor combineren we het op CPPN gebaseerde AESHN neuro-evolutie algoritme [9] met de op CPPN gebaseerde morfologische evolutie aanpak van [3]. Dit wordt gedaan door hun voorgestelde CPPN-architecturen samen te voegen tot één, waardoor één enkele CPPN zowel controller- als morfologiedefiniërende patronen kan produceren. Het belangrijkste verschil op het gebied van morfologie zijn de soorten kubusvormige bouwstenen die we beschikbaar stellen, wat in ons geval resulteert in starre morfologieën die zijn samengesteld uit verbonden en onvervormbare bouwstenen. Hoewel veel mogelijke typen bouwstenen door zowel mensen als geëvolueerde agents zelf kunnen worden ontworpen, zijn de belangrijkste typen die hier worden gebruikt:

- **Vulblok**: een algemeen blok dat wordt gebruikt om ruimte te "vullen", zonder een andere specifieke functie dan om verbinding te maken met andere blokken.
- **Hersenblok**: een uniek blok dat het substraat definieert. Dit is onderverdeeld in een binnen- en een buitengebied. Terwijl het binnenste gebied is gereserveerd voor verborgen neuronen die zijn ontwikkeld door het AESHN algoritme, kan het buitenste gebied worden gebruikt om door de gebruiker gedefinieerde invoer- en uitvoerneuronen te plaatsen.
- **Gewrichtsblok**: een blok dat een beweegbare verbinding bevat. De parameters die de vrijheidsgraden van het gewricht beschrijven, worden geëxtraheerd uit CPPN-geïnduceerde patronen. Dit blok herbergt zowel input- als outputneuronen. Terwijl de inputneuronen de huidige oriëntatie en positie van het blok ten opzichte van het hersenblok[2] krijgen, besturen de outputneuronen het beweegbare gewricht.

Hoewel het oorspronkelijke idee was om een *sensorblok* op te nemen waarmee het wezen de perceptie van zijn omgeving zelf kan evolueren, hebben we de initiële complexiteit tijdens het experimenteren (cfr. Paragraaf V) verminderd door de sensoren aan het hersenblok te bevestigen en de bijbehorende observaties aan de controller te leveren via door de gebruiker gedefinieerde inputneuronen.

Om een compleet wezen uit zijn CPPN-codering te halen, bouwen we eerst de morfologie door de CPPN op te vragen voor elke voxel-locatie binnen een doosvormige ruimte (gecentreerd rond de oorsprong), vergelijkbaar met het proces geïllustreerd in Figuur 1. Dit geeft ons op zijn beurt alle informatie die nodig is om de op ANN gebaseerde controller te bouwen met behulp van het AESHN-algoritme: een substraat (het hersenblok) en de locaties van zowel de morfologie-afhankelijke als door de gebruiker gedefinieerde input- en outputneuronen.

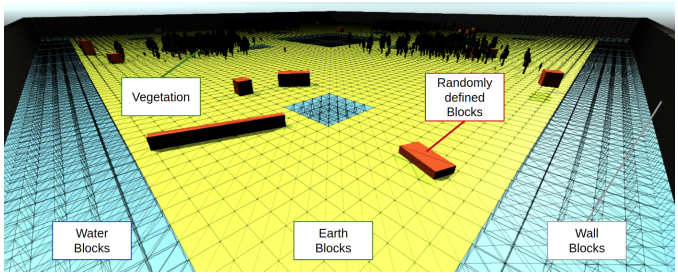
IV. Een uitgebreider Virtueel Ecosysteem

Naast een nieuwe aanpak die de complete evolutie van wezens beter toelaat, introduceren we een meer omvattende omgeving waarin dat evolutionaire proces kan plaatsvinden. Deze omgeving is een virtueel ecosysteem gemaakt in Unity [16], dat fundamenteel de ontbrekende componenten bevat van de virtuele werelden waarin evolutie tot nu toe is bestudeerd (cfr. Paragraaf I). Een voorbeeld van een instantiatie van het virtuele ecosysteem wordt getoond in Figuur 2.

Bij het ontwerpen van de omgeving hanteren we een bottom-up aanpak die het hele virtuele ecosysteem (en zijn bewoners) definieert door middel van modulaire bouwstenen. Het belangrijkste idee is dat een dergelijke modulaire wereld het mogelijk maakt om complexe omgevingsdynamiek gemakkelijker te introduceren, zoals steeds veranderende landschappen doordat blokken worden verplaatst of de eigenschappen van

[2]Dit is geïnspireerd door een natuurlijk effect dat kinesthesie wordt genoemd [15].

de ingevoegde blokken stochastisch te veranderen. De bouwstenen die hier worden gebruikt, zijn *aarde-*, *water-*, *muur-* en willekeurig gedefinieerde blokken. Stukken aardeblokken vormen het land waarop de kunstmatige organismen leven, terwijl stukken waterblokken de 'oceanen' vormen tussen die stukken land. Wandblokken kunnen worden gebruikt om een zichtbare grens van de omgeving te creëren, naast het plaatsen van ondoordringbare barrières om delen van de bewonende bevolking te isoleren. Van dergelijke ondoordringbare barrières is bekend dat ze de opkomst van soorten stimuleren [17]. Het doel van de willekeurig gedefinieerde blokken wordt hieronder besproken.



Figuur 2: Een gerenderde (wireframe) weergave van een voorbeeld van een virtueel ecosysteem. Dit is een vrij kleine en eenvoudige omgeving, aangezien deze slechts uit 102×102 blokken bestaat en een plat oppervlak heeft. Alle beschikbare bloktypes worden getoond en geannoteerd.

A. *(Inter)acties*

De beperking van interacties in veel werken komt voort uit het gebruik van een vooraf gedefinieerde of "hard-coded" reeks acties voor de wezens. Met andere woorden, een beperkt aantal acties beperkt het scala aan interacties. De behoefte aan zo'n vooraf gedefinieerde reeks acties werd verlicht door te vertrouwen op de fenotypische vrijheid van wezens die methoden zoals de voorgestelde OCRA-benadering het evolutionaire proces verlenen. Deze fenotypische vrijheid, veroorzaakt door de gelijktijdige evolutie van zowel schepselmorfologie als controller, introduceert een theoretisch onbeperkt aantal interacties tussen wezens. Het belangrijkste inzicht hierachter is dat de evolutie van wezens er nu toe kan leiden dat wezens deze acties en interacties zelf ontdekken. Bovendien opent het combineren van deze fenotypische vrijheid met een modulair virtueel ecosysteem (dat is samengesteld uit veranderlijke bouwstenen) een geheel nieuwe waaier aan interacties tussen het wezen en zijn omgeving. Eén type omgevingsblok, het *willekeurig gedefinieerde blok*, is speciaal voor dit doel aangewezen. De willekeur heeft betrekking op de geometrie, d.w.z. de grootte en vorm. Bij initialisatie van de omgeving wordt een door de gebruiker gedefinieerd aantal van dergelijke willekeurig gegenereerde blokken ingevoegd. Deze kunnen vervolgens door de wezens worden gebruikt voor alles wat ze nodig achten, zoals het mogelijk bouwen van "barrières" om "roofdieren" af te weren.

Niettemin zijn sommige basisacties in eerste instantie vooraf gedefinieerd om de complexiteitsdrempel van wezens die in het virtuele ecosysteem overleven te verlagen. Deze omvatten *eten*, *drinken*, *aanvallen* en *paren*. Terwijl eten en drinken triviale betekenissen hebben, maken de aanvallende en parende acties het mogelijk om twee belangrijke (inter-wezen) interactie-gerelateerde concepten op te nemen: *predatie* (wezens kunnen elkaar doden en opeten) en *seksuele reproductie*. Seksuele reproductie vormt de belangrijkste strategie om wezens hun collectieve doel te laten bereiken (het handhaven van een minimale populatiegrootte), naast het toestaan van een "verticale" mix van genetisch materiaal (nakomelingen dragen gemuteerde en gekruiste genen van hun ouders voort).

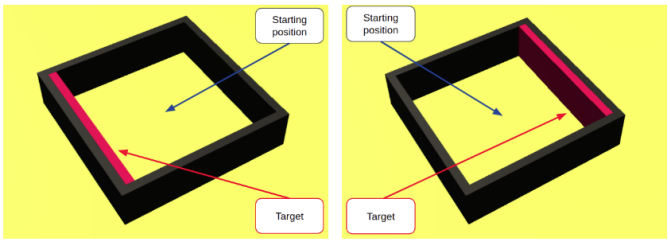
B. *Energie*

Binnen dit virtuele ecosysteem hebben wezens drie soorten energie: (1) *gezondheids-energie*, (2) *voedsel-energie* en (3) *drank-energie*. De maximale hoeveelheid energie die een wezen kan opslaan is voor al deze soorten gelijk en schaalt met het aantal bouwstenen waaruit zijn morfologie is opgebouwd (morfologische complexiteit). De toestand van alle drie deze energietypes wordt geleverd als extra input voor de controller van het wezen, waardoor het zijn gedrag kan veranderen afhankelijk van zijn energietoestand.

Om te overleven, moet een virtueel wezen voorkomen dat zijn gezondheidsenergie nul bereikt, wat kan gebeuren door " uithongering", " uitdroging" of aangevallen te worden door andere wezens. Terwijl verhongering optreedt wanneer de voedsel-energie van een wezen nul heeft bereikt, treedt uitdroging op wanneer de drank-energie nul heeft bereikt. Beide verminderen onafhankelijk van elkaar de gezondheids-energie met een constante snelheid, zodat wanneer beide gelijktijdig plaatsvinden, deze snelheden opgeteld worden. Gezondheidsenergie wordt geregenereerd wanneer het energieniveau van eten en drinken beide boven nul ligt. Hierbij worden deze twee soorten energie omgezet in gezondheidsenergie, waardoor hun eigen niveaus dalen.

Een wezen sterft wanneer het zijn maximale levensduur heeft bereikt (levensduur schaalt met morfologische complexiteit) of als zijn gezondheidsenergie nul bereikt. Dit biedt echter de mogelijkheid om dood te gaan met behoud van relatief hoge energiewaarden voor eten en drinken (bijvoorbeeld als de gezondheidsenergie door een aanval tot nul werd gedreven). De resterende voedsel- en drankenergie van een wezen blijft opgeslagen in zijn karkas, zodat andere wezens (bijvoorbeeld de aanvaller) het kunnen consumeren. Dit mechanisme maakt de interacties tussen roofdier en prooi heel natuurlijk. Waar voedsel-energie kan worden aangevuld door het eten van *vegetatie* (hieronder besproken), wordt drank-energie aangevuld door te drinken uit waterblokken. Beide soorten energie kunnen tegelijkertijd worden gewonnen door een karkas te eten.

Zowel de energie van eten als drinken raken uitgeput met een constant 'metabolisme', omdat wezens anders hun volledige leven zouden kunnen leven zonder actie te ondernemen om te overleven. Voedsel-energie wordt echter extra uitgeput door acties te ondernemen. De hoeveelheid energie die wordt verbruikt door *reproductie*, *aanvallen* en morfologische acties, d.w.z. gewrichtsbewegingen, schaalt met de complexiteit van zowel de morfologie (hoeveelheid bouwstenen) als de control-



(a) Trial 1 (b) Trial 2

Figuur 3: De Red-Wall omgeving, gemaakt in Unity [16]. Binnen deze omgeving wordt elk wezen geëvalueerd tijdens twee opeenvolgende proeven waarin het de doelmuur (rood) moet bereiken. Zoals getoond, is de locatie van de doelwand verschillend binnen elk van deze twee proeven. Om deze omgeving op te lossen, moeten de wezens evolueren naar een correct gebruik van hun sensorobservaties.

ler (hoeveelheid neuronen en verbindingen). Hoewel dergelijke acties er bijgevolg toe kunnen leiden dat de hongerdood sneller optreedt, kunnen de wezens, als ze de juiste acties ondernemen (bijvoorbeeld op weg naar eten en drinken), hun levensduur verlengen door te reageren op de druk van hun metabolisme.

C. Vegetatie

Naast de virtuele wezens herbergt het gepresenteerde virtuele ecosysteem nog een ander type virtueel organisme dat onderhevig is aan evolutie, hoewel op een veel lager niveau van complexiteit. Naast de karkassen van andere wezens, dient dit organisme als de belangrijkste voedselbron van de virtuele wezens en wordt het voorgesteld als stochastisch verspreide vegetatie. De belangrijkste interactie tussen deze virtuele organismen is dat wezens vegetatie kunnen eten om hun voedselenergie aan te vullen. Bovendien komt een extra reeks interacties voort uit het vermogen van wezens om vegetatie te verplaatsen. Deze interacties, gecombineerd met de genetische variatie die wordt veroorzaakt door de evolutie van beide virtuele organismen, dienen als een extra stimulans voor de evolutie van beide organismen.

V. Experimenten

Deze sectie bespreekt een van de belangrijkste experimenten die is uitgevoerd op ons pad naar het valideren van de voorgestelde OCRA aanpak om complete wezens te ontwikkelen. Er zijn ook veel andere experimenten uitgevoerd, maar deze zijn voor de beknoptheid uitgesloten.

A. Evolutie van Perceptie

Gezien de positieve validatie van de OCRA aanpak om volledige wezens te evolueren, die in staat zijn om te bewegen[3] en naar bepaalde doellocaties te bewegen in een eerdere reeks experimenten, ging de volgende reeks experimenten over het evalueren van het vermogen om wezens te evolueren die hun omgeving op de juiste manier waarnemen. Om dit te doen, hebben we een eenvoudige omgeving gecreëerd waarin het wezen zijn sensorobservaties moet gebruiken om een doel te lokaliseren en ernaartoe te bewegen tijdens twee opeenvolgende proeven. Deze omgeving kreeg de naam “Red-Wall” en wordt verder geïllustreerd in Figuur 3. De *fitness* van een wezen in elke proef is 1 als het het doel bereikt, met een extra factor die de fitness verhoogt op basis van het aantal resterende tijdstappen (waardoor snellere wezens worden aangemoedigd). Als het wezen het doel niet kon bereiken binnen een proef, wordt de fitness voor die proef berekend op basis van hoeveel dichter het bij het doel kwam in vergelijking met de startlocatie (0 is gelijk aan niet dichterbij zijn gekomen en $0,99$ bijna aanraken). De totale fitness van een wezen is dan gedefinieerd als het minimum van zijn fitness van de twee proeven, waardoor onbeduidende of “lucky” wezens onmogelijk een hoge fitness krijgen.

Hoewel het oorspronkelijke de bedoeling was om *sensorblokken* te laten evolueren binnen de morfologie van het wezen, d.w.z. de OCRA aanpak ook de plaatsing van sensoren te laten evolueren, verminderen we de complexiteit door de sensoren manueel aan het hersenblok te bevestigen. De bijbehorende waarnemingen worden via manueel gedefinieerde inputneuronen naar de controller gevoerd. De sensoren die hier worden gebruikt, zijn *raycast-sensoren*, d.w.z. sensoren die stralen in de omgeving uitsturen die het type object waarmee ze in botsing komen teruggeven.

Experimentele Setup

CVT-MAP-Elites [18] werd gebruikt als het onderliggende raamwerk om het evolutionaire proces te sturen. De gebruikte “behavior descriptor” bestond uit (1) een aantal richtingssteekproeven, dwz steekproeven die aangeven in welke richting het schepsel op bepaalde tijdstippen bewoog, deze vertegenwoordigen het traject van het wezen en (2) het (genormaliseerde) aantal opvul- en gewrichtsblokken binnen elk kwadrant van de morfologie van het wezen. De archiefgrootte van de elites was ingesteld op 100. De populatiegrootte was ingesteld op 128. Om de betrouwbaarheid van de resultaten te vergroten, hebben we acht keer dit experiment uitgevoerd. Elke dergelijke run werd uitgevoerd op een computer met vier *Intel Xeon E5-2650 v2 @ 2.60 GHz* CPUs, waarbij multiprocessing werd gebruikt om maximaal te profiteren van de 32 cores die beschikbaar waren.

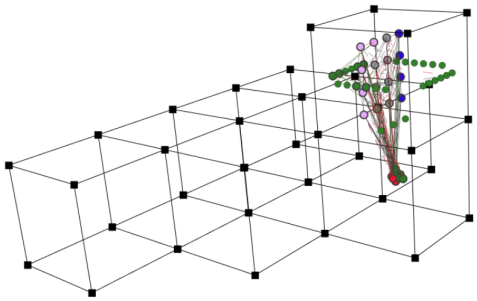
Resultaten

Drie van de acht runs die we hebben uitgevoerd, waren in staat om wezens te evolueren die in staat waren om het doel in beide proeven te bereiken. Hoewel het niet verwonderlijk is dat niet alle runs tot even succesvolle resultaten leiden, werd aangetoond dat hun falen voortkwam uit een gebrek aan genotypische (CPPN) complexiteit. Aangezien alle mutaties stochastisch plaatsvinden, is dit een natuurlijk artefact van het evolutionaire proces.

Een schematisch overzicht van het best presterende wezen[4] over alle runs wordt getoond in Figuur 4. Dit wezen werd na 392 generaties geëvolueerd (wat gelijk is aan 50176 evaluaties gezien de populatiegrootte van 128) en behaalde een fitness

[3]Een video die de beweging van een van de geëvolueerde wezens laat zien, is beschikbaar op: https://www.shorturl.at/dmoCJ

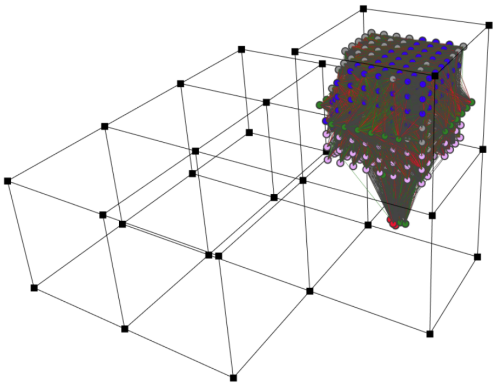
[4]Een video die de evaluatie ervan demonstreert, is beschikbaar op: https://www.shorturl.at/dvR01



Figuur 4: Schematische visualisatie van het best presterende wezen van het "Evolving Perception"-experiment. De morfologie van dit wezen is samengesteld uit 7 vulblokken en 1 gewrichtsblok onder het hersenblok. De controller die binnen het hersenblok is geëvolueerd, heeft in totaal 7 standaardneuronen (grijs), 4 recurrente neuronen (blauw), 5 neuromodulerende neuronen (roze) en 171 verbindingen. Neuromodulerende neuronen veranderen de verbindingsgewichten van andere (niet-neuromodulerende) neuronen afhankelijk van hun eigen activering, waardoor een hersengebied (bijvoorbeeld waar sensorische informatie wordt verwerkt) invloed kan uitoefenen op het leren van een ander gebied (bijvoorbeeld waar beweging wordt bepaald).

van 1.634. Deze fitness betekent dat het beide proeven met succes heeft voltooid, waarbij het in de "slechtste" proef slechts 36% van het maximaal aantal tijdsstappen vereiste. De run die dit wezen evolueerde, evolueerde ook 187 andere wezens die de omgeving hebben opgelost, verdeeld over 15 verschillende morfologische configuraties. Naast deze run evolueerde een andere run wezens met vergelijkbare morfologieën en controllercomplexiteiten, maar met ander bewegingsgedrag[5]. De laatste succesvolle run onderscheidt zich van de andere runs door richting veel hogere controllercomplexiteiten te evolueren, tot ongeveer 400 verborgen neuronen en 170000 verbindingen. Hoewel de controllers veel complexer waren, bleven de morfologieën en behaalde fitness scores vergelijkbaar met die van andere runs. Een voorbeeld wezen van deze run wordt getoond in Figuur 5. Interessant om op te merken is dat de controllers die binnen deze run zijn geëvolueerd, een modulaire structuur vertonen. Intuïtief is deze modulaire structuur opgebouwd uit een geheugenmodule (recurrente neuronen), een leermodule (neuromodulerende neuronen) en een verwerkingsmodule (standaardneuronen). Hoewel deze structuur een nogal interessant product is van het evolutionaire proces, naast het visueel aantrekkelijk zijn en enigszijns lijkend op de modulaire structuur van hersenen in de biologische wereld, wordt de omgeving waarin deze structuur geëvolueerd is als te eenvoudig beschouwd om de potentiële voordelen ervan te onderzoeken. Aangezien dit wezen echter al na 42 generaties werd geëvolueerd, kunnen er slechts maximaal 42 mutaties in het onderliggende CPPN-genoom hebben plaatsgevonden. Bovendien had de CPPN van het getoonde wezen geen verborgen neuronen. Dit suggereert dat de prestaties ervan voornamelijk afkomstig zijn van het krachtig coderingspotentieel van het onderliggende CPPN-genoom waaruit deze modulaire structuur is afgeleid.

[5]Een video die deze nogal eigenaardige, maar succesvolle beweging demonstreert, is beschikbaar op: https://www.shorturl.at/yHMR4



Figuur 5: Schematische visualisatie van een alternatief wezen dat geëvolueerd werd binnen het experiment "Evolving perception". De controller van dit wezen bevat 43 standaardneuronen (grijs), 357 recurrente neuronen (blauw), 143 neuromodulerende neuronen (roze) en in totaal 167409 verbindingen.

VI. Discussie

Waar de voorgestelde OCRA aanpak een grondige reeks experimenten onderging, vonden experimenten binnen het voorgestelde virtuele ecosysteem altijd plaats in sterk vereenvoudigde versies ervan. De belangrijkste reden voor de vereenvoudiging is de nieuwheid van de OCRA aanpak. Gezien zijn nieuwheid kunnen we niet verwachten dat het onmiddellijk de complexiteit van het complete virtuele ecosysteem op zich neemt en tot succesvolle resultaten leidt. Niettemin is het complete virtuele ecosysteem dat hier werd gepresenteerd volledig geïmplementeerd en klaar voor verdere experimenten[6].

Gezien de biologisch geïnspireerde opzet, inherente schaalbaarheid en uitbreidbaarheid, kan de gepresenteerde omgeving dienen als een testbed voor veel toekomstige evolutionaire werken binnen het kunstmatige levensdomein. We hopen dat sommige van deze werken de voorgestelde OCRA aanpak zullen bevatten, omdat we geloven dat deze omgeving zijn ware potentieel kan ontsluiten.

VII. Conclusie

Alhoewel we ons ervan bewust zijn dat de evolutionaire dynamiek op lange termijn zoals waargenomen in de biologische wereld nog ver buiten bereik is, geloven we dat de combinatie van de voorgestelde OCRA methodologie en het uitgebreidere virtuele ecosysteem ons in staat zal stellen belangrijke nieuwe inzichten te verwerven die kunnen helpen om de weg naar dit ambitieuze langetermijndoel te verlichten.

Referenties

[1] Nick Cheney, Josh Bongard, Vytas SunSpiral, and Hod Lipson. Scalable co-optimization of morphology and control in embodied machines. *J. R. Soc. Interface*, 15(143), June 2018.

[2] Antoine Cully, Jeff Clune, and Jean-Baptiste Mouret. Robots that can adapt like natural animals. *CoRR*, abs/1407.3501, 2014.

[6]Het virtuele ecosysteem, en alle andere code die in dit proefschrift is ontwikkeld, is beschikbaar op: https://github.ugent.be/dmarzoug/thesis

[3] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. *SIGEVOlution*, 7(1):11–23, August 2014.

[4] Nicholas Cheney, Josh Bongard, Vytas Sunspiral, and Hod Lipson. On the difficulty of Co-Optimizing morphology and control in evolved virtual creatures, 2016.

[5] B Weel, E Crosato, J Heinerman, E Haasdijk, and A E Eiben. A robotic ecosystem with evolvable minds and bodies. In *2014 IEEE International Conference on Evolvable Systems*, pages 165–172, December 2014.

[6] Allan Zhao, Jie Xu, Mina Konaković-Luković, Josephine Hughes, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. RoboGrammar: graph grammar for terrain-optimized robot design. *ACM Trans. Graph.*, 39(6):1–16, November 2020.

[7] Kenneth O Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genet. Program. Evolvable Mach.*, 8(2):131–162, June 2007.

[8] Sam Kriegman, Douglas Blackiston, Michael Levin, and Josh Bongard. A scalable pipeline for designing reconfigurable organisms. *Proc. Natl. Acad. Sci. U. S. A.*, 117(4):1853–1859, January 2020.

[9] S. Risi and K. Stanley. An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. *Artificial Life*, 18:331–363, 2012.

[10] Pablo Reyes and María-José Escobar. Neuroevolutive algorithms for learning gaits in legged robots. *IEEE Access*, 7:142406–142420, 2019.

[11] Tim Taylor. Evolution in virtual worlds. *CoRR*, abs/1710.06055, 2017.

[12] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, 2002.

[13] Kenneth Stanley, David D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15:185–212, 02 2009.

[14] Daniel Richards and Martyn Amos. Evolving morphologies with CPPN-NEAT and a dynamic substrate. *Artificial Life Conference Proceedings*, 26:255–262, July 2014.

[15] Jia Han, Gordon Waddington, Roger Adams, Judith Anson, and Yu Liu. Assessing proprioception: A critical review of methods. *Journal of Sport and Health Science*, 5(1):80–90, 2016.

[16] Unity Technologies. Unity.

[17] Larry Yaeger. Computational genetics, physiology, metabolism, neural systems, learning, vision, and behavior or polyworld: Life in a new context. 03 1995.

[18] Vassilis Vassiliades, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. Using centroidal voronoi tessellations to scale up the multidimensional archive of phenotypic elites algorithm. *IEEE Transactions on Evolutionary Computation*, 22:623–630, 2018.

Vulgarising Summary

In the domain of Artificial Life, researchers examine systems related to natural life and its processes through the use of simulations. The ultimate goal being the creation of intelligent life as encountered in our biological world. To do so, researchers draw inspiration from the process that has brought rise to intelligence in our biological world: evolution.

Evolution in nature can be described by the following observations:

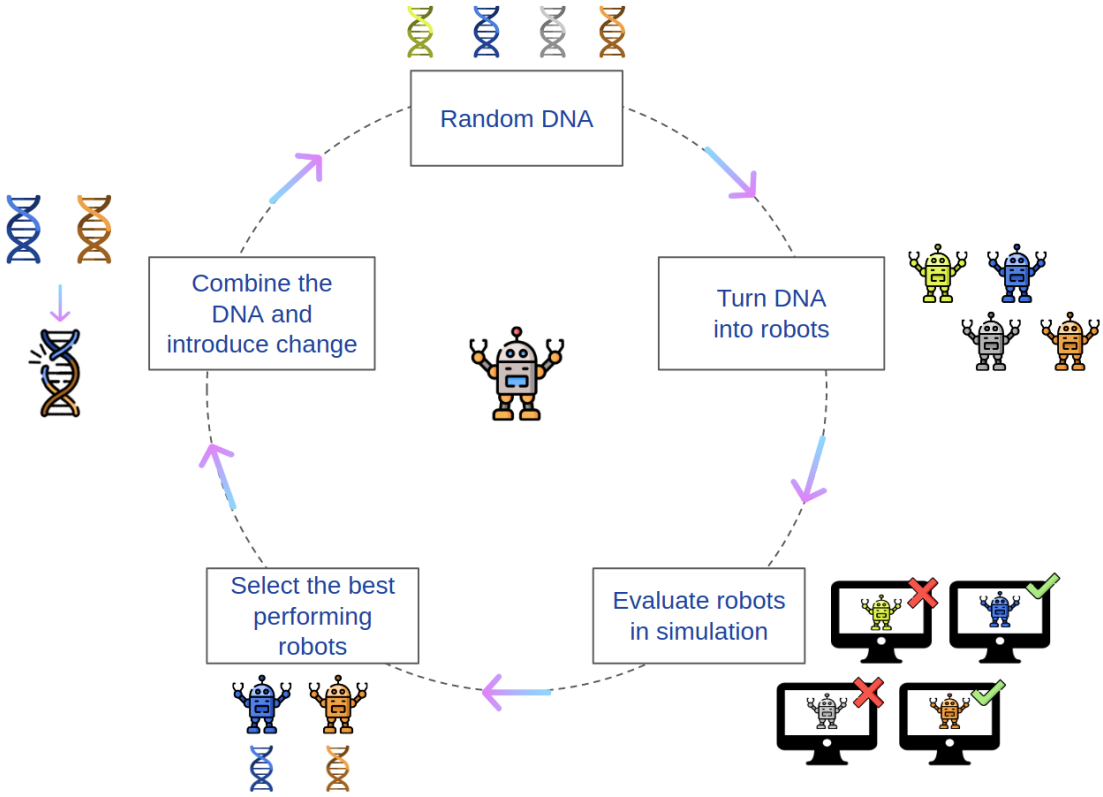
- Internally, animals are defined by some genetic encoding, most commonly known as 'DNA'.
- When reproducing, animals create children that carry on their DNA. However, they do not reproduce exact copies of themselves. The DNA of the child is a slightly modified version of the DNA of its parents. In other words, there is variation in between generations. Such changes in DNA can for example lead to a child being taller than its parents.
- Based on the environment in which these animals live, some characteristics are more favourable and allow certain animals to survive more easily. Say for instance that the only food source in a given area is the fruit growing on tall trees. Well, the child that had the 'luck' of being taller than its parents will more easily reach this fruit and therefore survive more easily. Consequently, that child will be able to reproduce more often than others. Its own children will then carry on this favourable DNA. This is commonly known as 'natural selection'.

Evolution can thus in a way be seen as an optimization process that spans many generations. Over time, animals (and all organisms in general) become more fit to their environment, because of the random changes in their genes that sometimes result in having an advantage

over others. Animals that have this advantage, are more likely to survive and cause that 'lucky' genetic change to be carried on.

Based on this insight, evolutionary principles have been applied for quite some time now within computer science. In computer science, many problems that researchers try to solve are of the optimization kind. These problems often have many possible solutions, but only a few are truly optimal ones. A typical example is designing the 'best possible robot' for some given task.

Well, this can be done automatically by applying the evolutionary principles. To do so, we first have to devise some description, a digital kind of DNA if you will, that can represent such a robot. Next, we can randomly create many instances of that DNA, turn them into actual robots and evaluate them on the task at hand. Some will do better, while others will do worse. We then pick the best performing ones out of the group and make them the parents of the next generation of robots. To create that next generation, we mingle the DNA of these selected parents and insert some random changes. This new collection of DNA instances is then again turned into actual robots and the process repeats itself. This goes on until we find a robot (or multiple robots) that are capable of solving the given task. The figure on the next page illustrates this process.



In the same way as evolution brought rise to intelligent life within our biological world, artificial life tries to do this in a (simplified) virtual twin of our world: a virtual ecosystem. Whereas in our biological world we speak of 'animals', in the virtual ecosystem we speak of 'virtual creatures'. Just like animals, these virtual creatures have a brain and a body.

While research concerning artificial life has already produced some promising results, including the emergence of complex interactions between such virtual creatures, the complexity and intelligence as seen in our biological world is still far beyond reach. While many things play part in this, our focus is the following:

- In natural evolution, the animals' brains and bodies are evolved simultaneously. However, when trying to do the same with virtual creatures (or robots) we encounter issues. Consequently, in work to date they are often not evolved truly simultaneously,

but rather by switching in between them from time to time. For instance, every 10 generations we switch from introducing variation in the bodies to introducing variation in the brains. Furthermore, these brains and bodies are often not represented with the same DNA but instead use two different kinds of DNA representations. However, this often results in the brain and body of creatures not being appropriately fit for each other.

- The virtual worlds in which the evolution of such creatures takes place, is often too simple. In other words, they just do not contain enough incentives to make the evolutionary process result in the level of intelligence and complexity that is desired.

In this thesis, we addressed these issues by:

- Introducing a new technique that better allows for the simultaneous evolution of creature brain and body. This is done by, just like in our natural world, representing both the brain and the body of these creatures using only a single genetic encoding. In other words, the complete creature is now described by just a single type of DNA. This means that when we evolve that DNA, both a creature's brain and body are evolved together.

- Introducing a more comprehensive environment in which the evolution of these virtual creatures can take place. This environment is a virtual ecosystem that is composed out of many small blocks[1], which the creatures can move around and interact with. The key idea behind this design is that it more easily allows for complex interactions to occur, both between creatures and between the creature and its environment. The virtual ecosystem therefore heavily relies on the improved technique of evolving virtual creatures discussed above. Given that the evolution of these creatures is now 'set free', in the sense that both their brain and body can now evolve to whatever seems fit (while

1. For those that are familiar, it somewhat resembles Minecraft.

still remaining functional), they can interact with each other and with this environment in many more ways than was possible in the environments that are available to date.

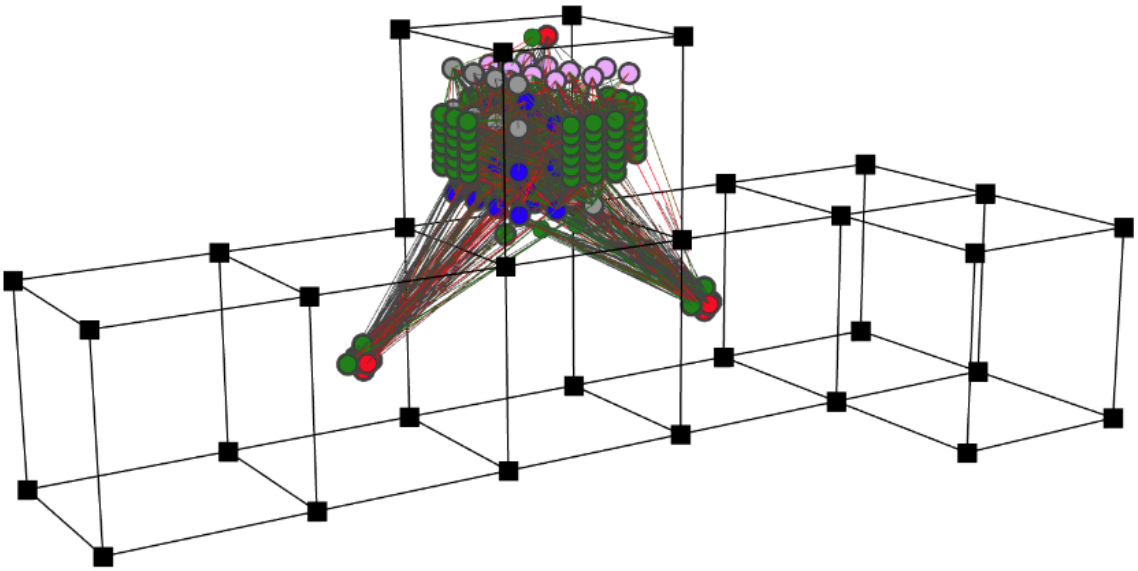
While many experiments were done to evaluate the new approach towards the simultaneous evolution of creatures' brains and bodies, many more are required to truly be certain of both its advantages and possible weaknesses. However, the experiments do provide a solid proof of concept of the proposed technique. Some experimental highlights are provided through videos:

- A creature with a fixed body, but an evolved brain that allows it to recognize food (plants), move towards it and eat it. In other words, an evolved lawnmower:
  `https://www.shorturl.at/hFHT0`
- A completely evolved creature that can walk around: `https://www.shorturl.at/dmoCJ`
- A completely evolved creature that can 'see' (in the environment shown in the video, the creature was given the goal to first identify the red wall and then move towards it):
  `https://www.shorturl.at/dvR01`

Experiments regarding the virtual ecosystem however always used simplified versions of it. Given that our approach of evolving virtual creatures is quite new, we cannot expect it to immediately take on the full complexity of the complete virtual ecosystem. Nevertheless, the virtual ecosystem is fully created and it is our hope that it will serve as the environment of choice for many works to come.

Lastly, the figure below shows a schematic visualization of a completely evolved creature using our approach. The cool thing to notice is that its brain, the block that contains many circles (neurons) and lines (synapses), is connected to two other blocks. These blocks are its

'joints' and allow it to move. Although heavily simplified, this reflects the nervous system in real animals, which connects the brain to the rest of the body.



Preamble

We move our arms and legs purposefully without any mental effort. However, encoding this ability into robots has proven to be a very difficult task. This observation is a puzzle that has always fascinated me: why is the level of intelligence and embodiment that comes natural to us, so difficult to understand and to cast into computers and robots. To this day, research has mainly focused on how intelligence itself can be designed. However, I believe we should direct our focus on a more natural and familiar process that has already shown its capacity of producing intelligence. Indeed, I believe the answer to this puzzle lies right in front of us: evolution.

First and foremost, I am extremely grateful to my supervisors Prof. dr. Yvan Saeys and Prof. dr. ir. Francis wyffels for their continuous support and guidance. They gave me the intellectual freedom of investigating the things that seemed most interesting to me, while still guiding me throughout this work with their invaluable expertise.

Special thanks go to my counsellor dr. ir. Matthias Freiberger for the many reviews of my work. His insightful feedback did not only allow me to vastly improve my writing skills, it also pushed me to sharpen my thinking and brought my work to a higher level.

I would also like to acknowledge Andreas Verleysen and Victor-Louis De Gusseme for their interesting feedback and ideas during the many (sadly online) thesis meetings we had. In particular, I would like to thank Andreas for his help in improving my presentations.

I hold all of you in high regard and I am forever grateful for the time and energy you have spent in supporting me throughout this thesis. I could not have wished for a better team.

Admission to Loan

The author gives permission to consult this master dissertation and to copy it or parts of it for personal use. Every other use falls under the restrictions of the copyright, in particular concerning the obligation to mention explicitly the source when using results of this master dissertation.

-

De auteur geeft de toelating deze masterproef voor consultatie beschikbaar te stellen en delen van de masterproef te kopiëren voor persoonlijk gebruik. Elk ander gebruik valt onder de bepalingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit deze masterproef.

The author,

Dries Marzougui - June 1, 2021

CONTENTS

1 INTRODUCTION 1
1.1 Envisaged Contributions 6
1.2 Outline 7

2 RELATED WORK 8
2.1 Evolutionary Algorithms 8
2.1.1 Genetic Algorithms 9
2.1.2 Novelty Search 11
2.1.3 Quality-Diversity 14
2.1.4 CMA-ES 18
2.2 Evolution in Virtual Worlds 20
2.2.1 A Chronological Overview 20
2.2.2 PolyWorld 23
2.3 Compositional Pattern Producing Networks (CPPN) 29
2.4 Morphological Evolution 30
2.5 Neuroevolution 33
2.5.1 General Overview of the Neuroevolution Domain 34
2.5.2 NeuroEvolution of Augmenting Topologies (NEAT) 36
2.5.3 HyperNEAT 42
2.5.4 Evolvable-Substrate HyperNEAT 47

I EVOLVING VIRTUAL CREATURES

3 ONE CPPN TO RULE THEM ALL (OCRA) 60
3.1 Creature Morphology 61
3.1.1 Agent Space 61
3.1.2 Building Creature Morphology 63
3.2 Creature Brain 66
3.2.1 Brain Substrate 67
3.2.2 Node Placement 67
3.2.3 Modifications to the AESHN Algorithm 70
3.3 CPPN Architecture 71
3.3.1 CPPN Querying 74
3.3.2 Activation functions 75

4 EXTENDING THE MAP-ELITES ALGORITHM 79
4.1 CMA-ES infused CVT-MAP-Elites 81

5 EXPERIMENTS . . . 85
5.1 Validating the Neuroevolution Algorithm . . . 85
5.1.1 Cartpole . . . 86
5.1.2 MountainCar . . . 91
5.2 Towards Moving Creatures . . . 94
5.3 Towards Seeing Creatures . . . 110
5.3.1 Extrinsic Guidance Experiment . . . 110
5.3.2 An Informative Detour of Failures . . . 122
5.3.3 Reducing Complexity through Crawlers . . . 130

II THE VIRTUAL ECOSYSTEM

6 A VIRTUAL ECOSYSTEM IN UNITY . . . 150
6.1 Overview . . . 150
6.1.1 Simulation Flow . . . 153
6.1.2 Analysis . . . 155
6.2 Unity . . . 155

7 VIRTUAL ORGANISMS . . . 157
7.1 Vegetation . . . 157
7.1.1 Genes . . . 158
7.1.2 Energy . . . 160
7.2 Creatures . . . 161
7.2.1 Genes . . . 163
7.2.2 Energy . . . 165
7.2.3 Interactions . . . 167

8 EXPERIMENTS . . . 174
8.1 Validating and Debugging through Reinforcement Learning . . . 174
8.2 Validating the Neuroevolution Algorithm . . . 175
8.3 Incorporating OCRA . . . 183

9 CONCLUSION, IMPACT AND FUTURE WORK . . . 197
9.1 A Novel Approach Towards Complete Agent Evolution . . . 197
9.1.1 Impact . . . 200
9.1.2 Future Work . . . 200
9.2 Towards a More Comprehensive Virtual Ecosystem . . . 201
9.2.1 Impact . . . 204
9.2.2 Future Work . . . 204

A CPPN ACTIVATION FUNCTIONS . . . 226

CHAPTER 1

INTRODUCTION

Evolution sculpts both the body plans and nervous systems of agents together over time. By contrast in evolutionary robotics, the application of evolutionary optimization to include both the robot neural controllers and morphologies has proved to be problematic [1]. While recent successes have demonstrated the potential of effective optimization for the control policies of agents with fixed morphologies [2, 3, 4] or to a lesser extent the optimization of morphologies for agents with minimal and fixed control policies [5, 6, 7], the simultaneous optimization of the two has seen very limited success [8, 9].

Cheney et al. [9] hypothesize that traditional evolutionary algorithms are hindered in this setting primarily due to an effect of embodied cognition, in which an individual's body plan and brain have an incentive to specialize their behaviors to complement one another. This specialization makes improvements to either subsystem difficult without complementary changes in the other (a highly unlikely event given current algorithms) and thus results in an embodied agent which is fragile with respect to design perturbations. Later in [1], Cheney et al. provide additional insights by demonstrating the increased effectiveness of an algorithm that addresses this hypothesized co-dependence. By allowing for *'morphological innovation protection'*, which temporarily reduces selection pressure on recently morphologically changed individuals, they enable evolution some time to "readapt" to the new morphology with subsequent control policy mutations.

Following the hypotheses made by Cheney et al., this thesis proposes an alternative approach to address the current issues of simultaneous evolution of agent brain and morphology. Current state-of-the-art approaches [10, 11, 1, 12] separate the genetic encoding for morphology and controller as this enables more control over the optimization process (e.g.

iteratively switching between morphology optimization and controller optimization). However, we hypothesize that evolving a single genetic encoding simultaneously representing both incorporates complementary changes between them in a more natural way, as both brain and body are then derived from the same higher-order parameter space, or *metaspace*. Consequently, this should also lead to a higher level of embodied intelligence[1] [13] as both brain and body are more acutely attuned to one another. This establishes the first research objective of this thesis.

However, this would require a rather powerful genetic encoding. Based on an exhaustive study of related work within this domain, we put forth the *Compositional Pattern Producing Network* (CPPN) [14], as described in Section 2.3. CPPNs have already successfully shown their ability to separately encode functional robot morphologies [15, 16, 6] and controllers [17, 18, 19] by representing them as *geometrical patterns* within some predefined space. Within this novel methodology, the CPPN would be used to generate both morphology and controller defining patterns within the same geometric space. As the CPPN would then be evolved to learn an appropriate relationship between the defining patterns of both the morphology and controller, we hypothesize that mutations on the CPPN should naturally lead to complementary changes within these patterns.

Besides serving purpose in the evolutionary robotics domain, a novel methodology that mitigates the current issues of complete agent evolution would be beneficial for the artificial life (ALife) domain as well. While tightly linked, the main difference between these two domains lies in their goal. Whereas research within the (evolutionary) robotics domain is often concentrated on designing the best possible robotic solution for some given task, research within the artificial life domain leans more towards the philosophical side and concentrates

1. The field of embodied intelligence places an emphasis on the role of an agent's body in generating the desired behavior, thereby allowing a simplification of the controller.

more on studying systems related to natural life and its processes trough the use of simulations. The goal in ALife is thus not necessarily to design the best possible robot, but rather to engage our imagination and creativity, to expand our conception of life and intelligence and how these things might materialize [20]. In other words, to create *artificial life*. Consequently, evolutionary techniques establish an important component of ALife. One popular object of study within the ALife domain are a type of embodied agents, often referred to as *virtual creatures*. Granted that these are composed out of both a morphology and a brain, they establish an interesting subject to study the simultaneous evolution of both these components on. Within this thesis, virtual creatures are therefore designated as the entity on which the proposed evolutionary techniques will be applied.

Furthermore, an environment is required in which these virtual creatures can be evolved. Within our biological world, evolution has shown its capability of leading to remarkable results. A (simplified) virtual twin of our world accordingly forms a straightforward choice for an environment in which evolutionary experiments can be studied. Such a virtual twin is often referred to as a virtual ecosystem.

Although research concerning evolution in the domain of virtual worlds (virtual ecosystems being an important subset) has already led to some promising results, including the emergence of *species* and complex interactions between virtual organisms, it has not yet succeeded in reproducing the long-term evolutionary dynamics as observed in the biological world. In other words, no work has achieved an open-ended evolutionary dynamic involving a long-term, intrinsic drive for increased diversity and complexity of virtual organisms [21].

One plausible explanation is that the scale of these systems, both in terms of population sizes and duration of simulations, has simply not been large enough to date. However, T.

Taylor [21] argues that there are a number of reasons to believe that the poor evolvability is due not just to issues of scale, but also to some more fundamental problems with the way in which these systems have been designed. He suggests that these problems mainly originate from an overemphasis on the requirements for a Darwinian evolution process to the exclusion of other aspects of biological theory. This overemphasis often results in systems that reduce the whole evolutionary process to natural selection, thereby employing an explicit rule - a "fitness function" - to determine which individuals are allowed to reproduce. However, many works have already argued that such an explicit fitness function limits the open-endedness of evolution [22, 23, 24], thereby restricting its potential. The most common way to avoiding an explicit fitness function has been through means of *self-reproduction*, i.e. requiring organisms to build their own offspring rather than employing an extrinsic mechanism to decide which organisms can reproduce [21]. One such work that employs self-reproduction was introduced by T. Ray in [22]. In his paper, he explains:

> *Self-reproduction is critical to synthetic life because without it, the mechanisms of selection must also be predetermined by the simulator. Such artificial selection can never be as creative as natural selection. The organisms are not free to invent their own fitness functions. Freely evolving creatures will discover means of mutual exploitation and associated implicit fitness functions that we would never think of. Simulations constrained to evolve with predefined genes, alleles, and fitness functions are dead-ended, not alive.*
>
> - T. Ray (1991) [22]

Nevertheless, the processes of self-reproduction with heritable mutation and selection, by themselves, are believed to be insufficient to explain the open-ended evolution of diversity and complexity. More specifically, T. Taylor [21] indicates the following three major issues: (1) the lack of complex dynamics within the environment itself, (2) the restrictiveness of ecological interactions (both between organisms and between the organisms and their environment) and (3) the lack of mixing genetic material between individuals either "vertically"

(through sexual reproduction) or "horizontally" (the exchange of genetic material between unrelated organisms).

In line with the considerations of T. Taylor discussed above, the second research objective of this thesis concerns itself with building a more comprehensive virtual ecosystem that fundamentally incorporates the missing components of current systems described above. Therefore, we will draw inspiration from one of the fundamental works within this domain, namely *PolyWorld* [25] introduced by L. Yaegar and further discussed in Section 2.2.2. However, we will deviate from it by taking a bottom-up approach that defines the whole virtual ecosystem and its inhabiting creatures by means of building blocks. The key idea being that such a modular world allows to more easily introduce complex environmental dynamics, such as ever-changing landscapes due to blocks being moved around or (stochastically) changing the properties of the inserted blocks. Moreover, the primary[2] food source of the inhabitants will itself be an organism, portrayed as vegetation, which undergoes evolution as well (although at a much higher level).

Furthermore, the potential range of interactions between the virtual creatures themselves and between the creatures and the environment will be opened up by incorporating the simultaneous evolution of both creature morphology and control (research objective one). Given that the morphologies are then no longer statically defined, we can steer away from these organisms having only a fixed set of "hard-coded" actions available to them as their evolved morphologies now define their possible actions. This combined with the modularity of their environment, could allow organisms to interact with that environment in ways not previously possible. For instance, a creature can use its evolved morphology to move blocks within the environment around, thereby changing the environment for both itself and other

2. The secondary food source being the cadavers of other inhabitants, thereby stimulating *predation* in the same way as proposed in PolyWorld [25].

inhabitants. By extent, this reduces the restrictiveness of inter-organism interactions witnessed in existing virtual ecosystems [25, 26]. Such inter-organism interactions will also be further stimulated through the possibility of sexual reproduction (thereby allowing a vertical mixture of genetic material between organisms), along with *predation*.

1.1 Envisaged Contributions

As its title implies, the ultimate goal of the research presented in this thesis is to evolve virtual creatures. Therefore, we indicated two research objectives. The aim of the first research objective is to address the current issues of simultaneous brain-body evolution through representing them both by a single genetic encoding. Within the evolutionary robotics domain, this would signify a powerful novel methodology towards the simultaneous creation of both robot morphology and control. Within the artificial life domain, this methodology would allow increasingly complex virtual creatures to be evolved. Subsequently, this would open up a whole new range of possible creature interactions and ethological behaviors to be investigated.

The second aim of the research presented in this thesis is to take a next step towards unshackling the full potential of virtual evolution by producing a more comprehensive environment in which that evolution may take place. This environment is a modular virtual ecosystem that fundamentally incorporates important stimuli for increased diversity and complexity of virtual organisms, which are believed to be lacking in the virtual ecosystems available to date [21].

Although we are aware that the long-term evolutionary dynamics as observed in the biological world are still far beyond reach, we believe that the combination of the two objectives presented here will allow us to gain important new insights that may help to illuminate the

path towards this ambitious long-term goal.

1.2 Outline

The next chapter begins with relevant background on evolutionary algorithms, evolution in virtual worlds, morphological evolution and neuroevolution. Part I then presents the work done related to the first research objective. Next, we introduce the more comprehensive virtual ecosystem in Part I. Lastly, we conclude in Chapter 9 by reflecting on what has been done, discussing the potential impact of the work presented here and suggesting possible directions for future work.

All code developed and used throughout this thesis, including the presented virtual ecosystem, is available at: `https://github.ugent.be/dmarzoug/thesis`

CHAPTER 2

RELATED WORK

In this chapter, the most important literature and published knowledge relevant to this thesis is described. In Section 2.1 we give an overview of more general techniques within the domain of *Evolutionary Algorithms* that were applied in this thesis. Next, in Section 2.2 we focus on the *Virtual Worlds* in which such evolutionary techniques have been applied to date, thereby denoting some foundational concepts concerning evolutionary processes in general. Section 2.3 then discusses the *Compositional Pattern Producing Network* (CPPN), a type of neural network that plays a central role within this thesis due to its usage as the *single genetic encoding* representing complete agents. The final two Sections 2.4 and 2.5 then respectively discuss fundamental works concerning *morphological evolution* and *neuroevolution*. In both of these sections, an emphasis is placed on works that employ the CPPN as the genetic encoding.

2.1 Evolutionary Algorithms

An important subset of the well-known Evolutionary Computing field, called *Evolutionary Algorithms* (EA), apply biologically inspired evolution mechanisms such as reproduction, mutation, crossover and selection to form a generic population-based optimization scheme [27]. Candidate solutions to the optimization problem play the role of individuals in the population. Depending on how these individuals are parametrically represented and depending on the nature of the particular problem at hand, the larger class of Evolutionary Algorithms can be subdivided into many subclasses. One of these sub-classes has a central role within this thesis, namely the *Genetic Algorithms* (GA) [28].

Section 2.1.1 gives a brief introduction to these Genetic Algorithms. Next, Section 2.1.2

introduces a new perspective on the selection mechanism by drawing inspiration from the characteristics of natural evolution and steering away from traditional fitness based search by introducing Novelty Search [24]. Based on insights gained by the Novelty Search approach, Section 2.1.3 discusses a recent addition to the evolutionary computation toolbox: Quality-Diversity algorithms [29]. Lastly, Section 2.1.4 introduces *CMA-ES* [30], one of the most popular algorithms within a different sub-class of the Evolutionary Algorithms: *Evolution Strategies* (ES).

2.1.1 Genetic Algorithms

As their name suggests, GAs represent candidate solutions or individuals within the population using a *genetic encoding*. The appropriate choice for a genetic encoding depends on the optimization problem at hand. The sole requirement a genetic encoding has, next to being able to represent a candidate solution, is that it must support the evolutionary operators described above. The relation between a genetic encoding (the *genotype*) and the candidate solution (the *phenotype*) it represents is named the *genotype-phenotype mapping*, according to genetic evolution terminology.

Once a genetic encoding has been established, an initial population of candidate solutions is generated randomly. The algorithm then repeats the following regenerational steps until a well performing solution has been found: (1) evaluate all generated candidate solutions based on a predefined quality metric (most often a *fitness* function, as discussed below), (2) select the most qualitative candidate solutions, (3) generate new candidate solutions, often called offspring, by applying the crossover and mutation operators on the previously selected candidate solutions, (4) replace the least-qualitative candidate solutions of the population with the newly created offspring. This general optimization scheme is further illustrated in Algorithm 1.

Algorithm 1 Canonical Genetic Algorithm

```
1: procedure GA(N)
2:     generation ← 0
3:     population ← GenerateRandomPopulation(N)
4:     scores ← Evaluate(population)
5:     while isNotTerminated(scores, generation) do
6:         parents ← Selection(scores, population)
7:         offspring ← Crossover(parents)
8:         population ← Mutate(offspring)
9:         scores ← Evaluate(population)
10:        generation ← generation + 1
11:    solution ← Fittest(population, scores)
12:    return solution
13: procedure EVALUATE(population)
14:    scores ← List()
15:    for genome ∈ population do
16:        candidate ← GenotypeToPhenotypeMapping(genome)
17:        score ← QualityMetric(candidate)
18:        scores.append(score)
19:    return scores
```

Next to an appropriate genotype, an important design choice lies in the quality metric used to select the "parents" of new offspring during each generation. While rather recently some alternatives have been proposed, such as the *novelty score* [24] discussed in Section 2.1.2, the *fitness function* is most common. A fitness function is a particular type of objective function that is used to summarise, as a single figure of merit, how close a given candidate solution is to achieving the target [31].

Given their genericity, GAs have been successfully applied in a wide range of use cases, spanning over many different domains. Two domains that are most relevant for the research presented here are *evolutionary robotics* [32] and *neuroevolution* [33]. Whereas evolutionary robotics targets the automatic design of (subsystems) of robots through evolution, neuroevolution focuses on the generation of artificial neural networks (ANN). Granted that fully

automated robots require controllers to steer their morphology, neuroevolutive algorithms are often employed within the evolutionary robotics domain. Morphological evolution and neuroevolution will be further discussed in Sections 2.4 and 2.5 respectively.

2.1.2 Novelty Search

One of the critical evolutionary mechanisms used in all genetic algorithms is the *selection* operator, an abstraction of *natural selection*. For a given population of genomes, it *selects* some of them to be used as the parents to create the offspring (through mutation and cross-over) that forms the population of the next generation. Therefore, it evaluates the genomes based on some score, which enables ranking them such that only the current best scoring genomes get *selected* and are further evolved. In genetic algorithms, the selection operator thus steers the evolutionary search towards better solutions for the given task.

Until rather recently, the score used by the selection operator was almost always the same, namely the *fitness* of the individual. Drawing inspiration from the *"Survival of the Fittest"* idea described by H. Spencer [34], the fitness score causes only the best performing individuals of the population to be selected and optimizing this can be thought of as an abstraction of the selection pressure seen in natural evolution. The implicit assumption made here is that such optimization of fitness is an *accurate* abstraction of the high-level process that allows natural evolution to discover its complex and astonishing products. However, a consensus in biology is building that the pressure to maximize fitness may not be responsible for complexity growth in natural evolution [35].

In the evolutionary computation domain it often occurs that by optimizing purely based on fitness, the search converges into deceptive local optima that might appear promising, but from which no local step in the search space leads to an improvement. One reason for

this is the usage of nonlinear models that result into non-convex fitness landscapes. While a good choice of fitness function can somewhat alleviate this issue, the problem is that they are often just a heuristic or simplification of the actual performance of the individuals, as devising the "perfect" fitness function is seldom possible. Consequently, fitness functions often have a notion of *deceptiveness*, i.e. there is no guarantee that they will correctly reward important intermediate steps or *stepping stones* in the search space that are necessary to reach the objective of the search. In this way, quite paradoxically, the mechanism used in evolutionary computation to encourage the evolution of the best possible solutions may hinder its own progress and actively misdirect search toward dead ends.

Stepping stones A nice example of an important stepping stone can be given in the robotic gait learning domain, in which the goal is to make robots learn how to walk. Just like for humans, a primary stepping stone here is to first learn how to fall over, as it allows to get a grasp of the dynamics of walking. The typical fitness function in this domain rewards individual solutions based on how far they were able to move the robot from its initial starting location. Consequently, an agent that just falls over will not receive a high reward as it was not able to move that much. Therefore, although it discovered an important stepping stone, it is wrongly assumed to be a total failure and is potentially discarded completely.

Novelty Search In order to address the issues of deceptive fitness functions, *novelty search* [24] provides a whole new perspective by dropping the fitness score completely. Instead, as its name implies, it employs a novelty score that rewards individuals whose functionality is significantly different (*novel*) from what has been discovered before. In this way it directly steers the evolutionary search towards discovering the crucial stepping stones necessary to solve a given task, ones that a fixed fitness function may fail to perceive as important. By rewarding individuals to be as different as possible from prior individuals, it creates the constant pressure to do something new. This, as it turns out, is quite a good stimulus to guide

evolutionary search [36, 37, 38, 29]. In some problems ignoring the goal thus outperforms looking for it, as sometimes the intermediate steps to the goal do not resemble the goal itself.

In the robotic gait learning example described above, novelty search would reward the falling agent with a higher novelty score, as falling over can be seen as novel behavior. This allows the evolutionary search to use that stepping stone as a jumping-off point for further evolution.

Calculating novelty The novelty score is a measure that describes how different an individual is from the already discovered individuals in terms of its functional behavior. In order to calculate it, we thus have to first design a numerical (and task dependent) *behavior descriptor*. In the gait learning task described above, this behavior descriptor could for example be a sequence of numbers that tracks the robot's location and center of gravity throughout the simulation.

The novelty of a newly generated individual is then computed with respect to the behaviors of an archive of past individuals and the current population. The aim is to characterize how distant the new individual is from the rest of the population and its predecessors in *behavior space* (the space of unique behaviors). A good novelty metric should thus compute the sparseness at any point in the behavior space. Areas with denser clusters of visited points are less novel and therefore rewarded less [35]. The sparseness of a point can for example be calculated by taking the average distance to the nearest k neighbours of that point in behavior space, where k is an experimentally determined parameter. Equation (2.1) shows this novelty metric $n(b_i)$ for an individual i with behavior descriptor b_i.

$$n(b_i) = \frac{1}{k} \sum_{j=0}^{k} d(i, j) \tag{2.1}$$

For additional information about the novelty search approach, I kindly refer the reader to the following webpage[1] which includes a great summary, discussion and visual demonstration.

2.1.3 Quality-Diversity

This section covers a recent addition to the evolutionary computation toolbox: Quality-Diversity (QD) algorithms. The main idea is that QD algorithms do not only search for a single set of local optima, but instead try to illuminate the whole search space. To do this, they draw inspiration from the novelty search approach (cfr. Section 2.1.2) and typically work in the behavioral (feature) space instead of the genotypic (parameter) space[2]. Within this behavior space, QD algorithms then attempt to find the best performing solution (or "elite") for every behavioral niche, even if the niche is not a peak in the fitness landscape. This allows them to provide a holistic view of how high-performing solutions are distributed throughout the search space [29]. As the name indicates, Quality-Diversity algorithms aim to provide a large set of solutions spanning many behavioral niches (*diversity*), while still optimizing them locally (*quality*).

At first sight, QD algorithms look a lot like multitask optimization, that is, independently solving an optimization problem for each combination of features [40]. The main insight behind the QD approach is that solving this set of problems is likely to be faster if they are all solved together than by independent optimizations [29]. Intuitively, it seems likely that high-performing solutions for similar behaviors will be similar, therefore sharing information between their optimizations can indeed be beneficial. Additionally, independent optimization would be especially wasteful in a black-box optimization context because a candidate solution that does not have the right behavior for the current optimization would be com-

1. http://eplex.cs.ucf.edu/noveltysearch/userspage/

2. Working in behavior space instead of the genotypic parameter space differentiates them from multimodal optimization algorithms [39]

pletely discarded, whereas it could be very useful for a different behavior's optimization.

The outcome of QD based optimization is thus a set of solutions, often called a "collection", "archive" or "map". This set of solutions is expanded, improved and refined iteratively during the optimization process by following the typical select-mutate-evaluate scheme). In this case, selection is done by randomly picking individuals from the current set of solutions stored in the archive. Each individual within this archive represents a different type of solution (behavior) and covers some region within the behavior space. In practice, every evolved solution will have some behavior descriptor which enables to locate it within behavior space. Only solutions with similar behavior will then compete with each other to be maintained in the archive and thereby compete for the right to represent the elite of that behavioral niche.

In essence, this means that a QD algorithm has to segment the (usually continuous) behavior space such that it can fill as many cells as possible within this segmented space with elites. Here, every cell corresponds to one behavioral niche.

MAP-Elites One of the simplest and most popular ways to implement this segmentation of the behavior space is by discretizing it into a n-dimensional grid, as shown in Figure 2.1 and Algorithm 2. This approach was originally introduced by the *"Multi-dimensional Archive of Phenotypic*[3] *Elites"* (or *MAP-Elites* in short) algorithm [41] and has been successfully applied to many domains. Examples within the robotics domain include: enabling robots to quickly adapt to damage [42, 3, 43], morphological designs for walking soft robots [41], controllers for robotic arms [41] and neural networks that steer simulated robots through mazes [44].

3. Phenotypic refers to the use of the behavioral space instead of the genotypic space as the working ground for the algorithm.

Algorithm 2 MAP-Elites algorithm, as presented in [29]

procedure MAP-ELITES($[n_1, ..., n_d]$) ▷ n_i is the user-defined discretization for behavior dimension i
  $A \leftarrow$ CreateArchive($[n_1, ..., n_d]$)
  for $i = 1...G$ **do** ▷ Initialization: G random candidate solutions Θ
    $\Theta \leftarrow$ RandomSolution()
    AddToArchive(Θ, A)
  for $i = 1...I$ **do** ▷ Main loop, I iterations
    $\Theta \leftarrow$ Selection(A)
    $\Theta' \leftarrow$ Variation(Θ)
    AddToArchive(Θ', A))
  return A
procedure ADDTOARCHIVE(Θ, A)
  $(p, b) \leftarrow$ Evaluate(Θ) ▷ b is the behavior descriptor of Θ, p is performance
  $c \leftarrow$ GetArchiveCell(b)
  if $A(c) == null$ or $A(c).p < p$ **then**
    $A(c) \leftarrow p, \Theta$

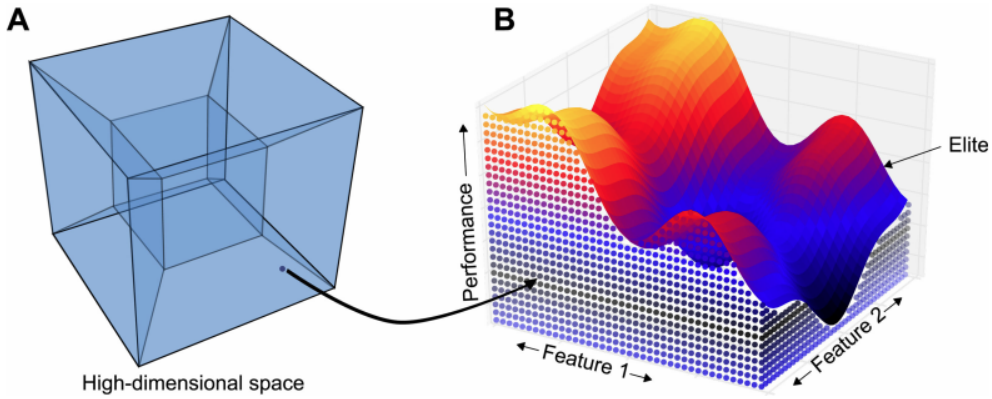


Figure 2.1: Illustration of the behavioral or feature space discretization of MAP-Elites, as given in the original paper [41]. The MAP-Elites algorithm searches in a high-dimensional space to find the highest-performing solution (elite) at each point in a low-dimensional behavioral space. This behavioral space is defined by the user of the algorithm and the illumination of the fitness potential of each area in within this space can provide interesting information. For example, MAP-Elites could search in the space of all possible robot designs (a very high-dimensional space) to find the fastest robot (a performance criterion) for each combination of height (feature 1) and weight (feature 2).

CVT-MAP-Elites Although the grid-based archive of MAP-Elites has various benefits, such as the conceptual simplicity and allowing qualitative evaluation (e.g. by visual inspection of a 2D MAP), it runs into trouble when trying to scale it to high-dimensional feature spaces. As each feature dimension is discretized into multiple (user defined) intervals, the total number of archive bins increases exponentially with the number of feature dimensions. For instance, for a 50-dimensional space and 2 discretizations per dimension, MAP-Elites creates an empty matrix of 1.13×10^{15} cells that requires 4 petabytes of memory (assuming 4 bytes for the pointer of each cell) [40]. Additionally, the increase in number of niches also results in a reduced selective pressure (the probability that some solution is selected for further evolution), which is known to degrade performance as the 'depth' of the evolutionary optimization decreases.

One solution to the scaling problem of the MAP-Elites archive was given by Vassiliades et al. in [45]. By drawing inspiration from methods of the computational geometry domain, they propose to use a centroidal Voronoi tesellation (CVT) [45] to divide the behavioral space in a fixed number of regions. The CVT technique is known to effectively partition high-dimensional spaces into well-spread geometric regions, making it an attractive solution for this issue. Figure 2.2 illustrates how this new method, appropriately named *CVT-MAP-Elites* differs from the original MAP-Elites.

Experiments show that this extension allows CVT-MAP-Elites to achieve the same performance irrespective of the dimensionality, in contrast to MAP-Elites [29, 46]. Next to a better partitioning of the behavioral space, the ability to explicitly set the number of niches k allows to have more precise control over the balance between diversity and performance. This in turn mitigates the issues of MAP-Elites concerning the reduced selective pressure.

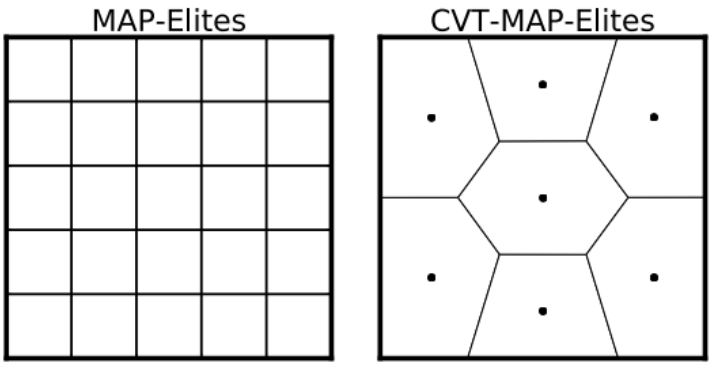


Figure 2.2: Illustration of the different behavioral space discretizations made by MAP-Elites and CVT-MAP-Elites, as given in [45]. MAP-Elites discretizes the behavioral space according to some pre-specified number of discretizations per dimension, causing the number of niches to grow exponentially with the number of dimensions. Consequently, it encounters issues when scaled up to high-dimensional behavioral spaces. In contrast, CVT-MAP-Elites uses a CVT to partition the behavioral space into k homogeneous geometric regions, where k is the pre-defined number of niches. In this example, MAP-Elites uses 5 discretizations per dimension, resulting into 25 niches, whereas CVT-MAP-Elites only uses $k = 7$ niches.

2.1.4 CMA-ES

The Covariance Matrix Adaptation Evolution Strategy [30], or CMA-ES in short, is a popular evolutionary algorithm for difficult non-linear non-convex black-box optimization problems in the continuous domain. As its name indicates, it is a type of *Evolution Strategy* (ES).

Vanilla ES In general, evolution strategies try to learn distribution parameters in such a way that vectors of real numbers sampled from that distribution optimize some objective function $f(x)$. To do this, it follows the typical evolutionary computation approach: (1) sample a population of candidate solutions based on the current distribution parameters θ, (2) evaluate the objective function on each of these solutions and (3) select the best subset of individuals (by some score measure, often fitness) and use them to update θ.

The canonical implementations most often use an n-dimensional isotropic Gaussian distribution, in which θ only tracks the mean μ and the standard deviation σ. Here, the standard

deviation σ accounts for the level of exploration: the larger the σ the bigger the search space in which we sample our offspring population.

CMA-ES The problem with the simple form of evolution strategy described above is that the σ of the previous step heavily influences the σ used in the current step. Consequently, the algorithm is not able to rapidly adjust the exploration space when needed (e.g. when the confidence level changes). CMA-ES fixes this problem by tracking pairwise dependencies between the samples in the distribution by iteratively updating an additional distribution parameter: the covariance matrix. Intuitively, the covariance matrix allows CMA-ES to cast a wider net of samples when the best solutions are far away or narrow the search space when the best solutions are nearby (cfr. Figure 2.3). For more details and a thorough mathematical description of CMA-ES, see [47].

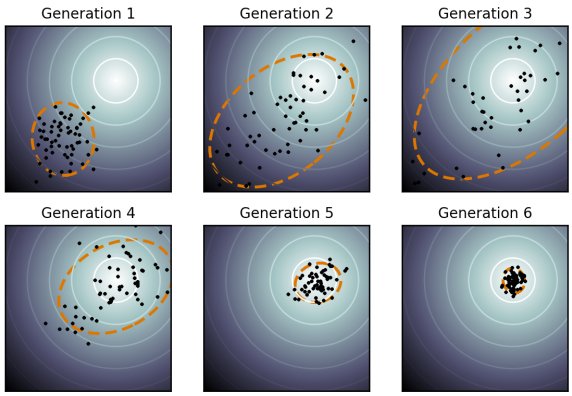


Figure 2.3: Illustration of a CMA-ES optimization run on a simple 2D problem as given in [48]. The solid lines of equal f-values depict the spherical optimization landscape. The population of samples (dots) shows how the underlying Gaussian distribution (dotted line) changes during optimization and alters its exploration space based on the iteratively updated covariance matrix.

Extensions Given its ease of application (quasi-hyperparameter-free) and decent convergence speed, CMA-ES is a quite popular method and has seen many successful applications [49, 50]. The major limitation of the original CMA-ES is its scalability to higher dimensions.

This is because learning a full covariance matrix introduces non-trivial algorithm internal cost and hence makes it a first class victim of the curse of dimensionality [51]. However, recent research has proposed many extensions to CMA-ES that address this issue, either by restricting the covariance matrix to the diagonal, to a low rank model or to a combination of both [52, 53, 54, 55].

2.2 Evolution in Virtual Worlds

Building further upon the introduction given in Chapter 1, this section discusses several important works regarding evolution in virtual worlds. Based on a great review written by T. Taylor in [21], Section 2.2.1 gives a chronological overview of the research done within this domain, thereby denoting some foundational concepts concerning evolutionary processes in general. Thereafter, Section 2.2.2 dives deeper into one work, named *PolyWorld* [25], which can be seen as one of the fundamental works within the domain of virtual ecosystems (a more biologically inspired subset of virtual worlds).

2.2.1 A Chronological Overview

The first ideas concerning evolution in virtual worlds date back to the late 1940s, when von Neumann became interested in the question of how complicated machines could evolve from simpler ones [56]. Specifically, he was interested in self-reproducing machines, i.e. machines that could build copies of themselves. He envisioned these machines being robust in the sense that they could withstand some types of mutation and pass these mutations on to their offspring. Such machines could therefore be subject to evolution [57]. To demonstrate his ideas, von Neumman developed a two-dimensional cellular automaton framework. This work can be seen as the first attempt to instantiate an evolutionary process in the context of a modern, digital computational framework [21]. One of his fundamental ideas, namely that of self-reproduction, is reflected in many of the later works by other researchers.

Although von Neumann established a system that had the *potential* for an evolutionary increase in complexity, he did not thoroughly address the question of where the *drive* for such an increase may arise from within the evolutionary system itself. One of such driving forces for increased complexity is for instance the interactions between machines. Some later works (1960s) of computational evolutionary systems did consider such *interorganism* interactions [58, 59, 60]. While exhibiting some interesting ecological and evolutionary dynamics, the attempts to evolve complex behaviors were met with limited success. This was explained by Conrad and Pattee by stating that variation and natural selection alone is not necessarily sufficient to produce a viable evolutionary process, even when embedded in the context of an ecosystem [60].

A while later in 1991, Ray introduced *Tierra* [22]. Being one of the most notable attempts towards a computational system capable of hosting an open-ended evolutionary process, this work studied the evolution of self-reproducing computer programs[4]. In this work, the "virtual world" came down to an initially empty block of computer memory into which a single seed program (written by Ray himself) was placed. This seed program then started to reproduce by copying itself one instruction at a time to a new location in memory. This copy (and subsequently its own copies) then continued this reproduction until the complete block of memory filled up. When the memory was full, the operator system handled the removal of older programs to make room for new ones. To introduce variation within the programs, random mutations to the instructions were probabilistically introduced during the copy operations. These mutations led to some interesting results such as the programs evolving to reproduce more quickly, by optimizing their own self-reproduction algorithm. However, the most interesting results were due to ecological interactions. For instance, the

4. These computer programs were written in a language based on modern assembly code

evolutionary process was observed to lead to *parasitism*, where short programs emerged that could only reproduce with the help of larger "host" programs. In response, other programs were evolved that became resistant to these parasites, undermining them to enforce their own reproduction. While impressive, each run of the system eventually stagnated, converging to some state in which only neutral variations occurred.

A couple of years later in 1994, L. Yaeger introduced *PolyWorld* [25], a complex two-dimensional *virtual ecosystem* containing evolving agents. Virtual ecosystems are a subset of the more general virtual worlds, wherein the environment draws inspiration from our natural world. The agents in PolyWorld are controlled by genetically determined neural networks and their goal is to survive within the ecosystem through collecting energy, fighting and mating. PolyWorld can be seen as a fundamental work within the domain of virtual ecosystems and will therefore be more thoroughly described in Section 2.2.2.

Around the same time, Sims introduced his approach towards evolution of creatures in a virtual world [61] and this is to this date still known as one of the most visually impressive works[5]. While he also evolved the creatures using a genetic description (this is further detailed in Section 2.4), he chose to use a three-dimensional virtual world featuring Newtonian mechanics. One of the reason why Sim's system produced such good results was that he modeled the physics of this three-dimensional environment accurately enough such that objects moved realistically when subjected to forces and torques. Hence, the graceful movements produced by many of his evolved creatures were due just as much to the accurately modeled physical environments as they were to the creatures' individual controllers [21].

To conclude this chronological overview of works concerning evolution in virtual worlds,

5. We kindly request the reader to take a look at the following video, showcasing this work: `https://www.youtube.com/watch?v=JBgG_VSP7f8`

we quote a paragraph from the future work section of Sim's paper [61] as it nicely lines up with the second research objective of this thesis:

"An additional extension to this work would be to simulate a more complex but more realistic environment in which many creatures simultaneously compete and/or cooperate with each another, instead of pairing off in one-on-one contests. Speciation, mating patterns, competing patterns, and even offspring production could all be determined by one long ecological simulation. Experiments like this have been performed with simpler organisms and have produced interesting results including specialization and various social interactions (Tierra, Polyworld)." - K. Sims 1994

2.2.2 PolyWorld

In essence, PolyWorld is an ecological simulator of a simple flat world, possibly divided up by a few impassable barriers and inhabited by a variety of organisms and freely growing "food" [25]. Its name is derived from the fact that all visible constituents in this world, including the organisms, are represented by simple polygonal shapes. The sole goal of these organisms or *"virtual creatures"* is to survive in this world through replenishing their energy and reproducing, such that their offspring can carry on their (albeit mutated) genes. Gaining energy can be done through searching and eating the food that grows around in the environment. This energy is necessary to perform actions, as each such action requires and expends energy. The goal of the impassable barriers is to stimulate *speciation* (genetic diversity), as these allow to partially or completely isolate populations of organisms. Figure 2.4 presents a screenshot taken from the PolyWorld ecological simulator.

PolyWorld's Virtual Creatures The organisms or virtual creatures in PolyWorld are completely defined by a set of (numerical) genes. The genes completely encode both the creature's neural architecture and physiology.

The neural architecture refers to the creatures' neural network brains. These brains receive vision as input and employ Hebbian Learning to change the connection weights during lifetime. The outputs of this brain fully determine the creatures' actions. While one could say that PolyWorld's vision based neural systems are quite elegantly designed, they will not be discussed here given that the focus of this section is on the dynamics of the virtual ecosystem.

The genes related to physiology include a creature's size, strength, maximum speed, life span, next to reproduction related concepts such as mutation rate, the number of crossover points and the fraction of energy that is passed to offspring. The specific values of these genes directly affect the creature's metabolism and traits. A larger creature for instance expends more energy when performing actions such as moving, but in turn is able to store more energy than smaller creatures. Another example can be given by the strength gene, as this also affects both energy expenditure and advantage in a fight (which is discussed later). Many more such examples concerning the dynamics of genes exist and fundamentally they all come down to trade-offs between certain genetic configurations. This interplay between these opposing advantages is intended to produce niches in the fitness landscape, and based on the ever-changing genetic state of the population, these niches may change over time. As should be evident from the examples above, there is for instance a clear energy conservation benefit to being small and weak, yet there are clear predatory advantages to being large and strong.

Each creature has a predefined suite of possible actions, which can be summarized as follows:

- **Eating**: a creature's method for replenishing depleted energy. To be able to do so, the creature's position must overlap with a piece of food.
- **Mating**: a creature's method for reproducing. In order to reproduce, the creature must

overlap with another creature that also expresses its mating behavior. The outcome of the mating attempt may be effected by the miscegenation[6] function. The creature's "desire" to mate, i.e. the activation level of the neuron corresponding to this action, is mapped onto its blue color component (thereby allowing other creatures to spot potential mates by using their vision based input).

- **Fighting**: a creature's method for attacking another creature. To be able to do so, the creature's position must overlap with the target. This forms the main mechanism behind the predator-prey interactions described further below. The creature's "desire" to attack is mapped onto its red color component (thereby allowing other creatures to spot the potential danger that they are in).

- **Moving and turning**: a creature's method for forward and turning motions. The amount it will move forward or turn is proportional to the activation level of the corresponding neuron.s

- **Focusing**: a creature's method to control its vision related field of view, making it possible to survey most of the world in front of it or to focus more closely on certain regions.

- **Lighting**: a creature's method to control the brightness of certain regions on the front face of its "body". This allows for a simple form of visual communication. However, no evidence of the actual use of this form of communication has been discovered.

In PolyWorld, there are two classes of energy storage in each creature: *health-energy* and *food-value-energy*. Both of these are replenished by eating food and both are depleted by neural activity and by engaging in the various behaviors. However, when a creature is attacked,

6. The "miscegenation function" (so dubbed by Richard Dawkins) stochastically influences the likelihood of genetically dissimilar creatures producing viable offspring. The greater the dissimilarity, the lower the probability of successful reproduction.

only its health-energy is affected. If this health-energy drops below zero, the creature dies and its remaining food-value-energy is converted into a piece of food. This separation of health-energy from food-value-energy makes the *predator-prey* interactions quite natural, i.e. it is possible for a creature to be killed by having its health-energy driven to zero, while still maintaining a relatively high food value for the attacker [25].

Running the simulation A simulation is considered successful if and only if some number of species emerge with a *Successful Behavior Strategy (SBS)*; these are populations which are capable of sustaining their numbers through their mating behaviors and thus no longer require additional artificial creations to maintain a minimum amount of creatures. This allows to break down simulations in PolyWorld (and virtual ecosystems in general) into three main phases: (1) initialization, (2) pre-SBS and (3) post-SBS.

As one may expect, the first phase initializes the virtual ecosystem and evolutionary process by inserting an initial seed population. This seed population consists of (a user-defined amount of) creatures of which the genes are randomly generated.

The second phase then tries to stimulate the population to obtain an SBS by running in an "on-line Genetic Algorithm (GA)" mode (also known as Steady State GA), in which an ad hoc fitness function is employed. During this phase, a minimum number of creatures is guaranteed to populate the virtual ecosystem. If the number of creatures drops below this minimum, either a randomly generated creature or the offspring of two creatures stored in an archive of the fittest creatures ever seen may be inserted. This ad hoc fitness function rewards organisms for eating, mating, living their full life span, dying with reserve energies and simply moving [25].

Once an SBS has emerged[7], we move into the third phase. Here, the backbone GA is turned off and thus there is no fitness function except survival. This means that there is no longer any intervention by the system (or humans) and the evolutionary process is from now on "completely free".

While the simulation is running, the graphical interface of PolyWorld includes some graphic displays of the time histories of certain quantities of interest, such as (1) population sizes, (2) the past maximum, current maximum and current averages of the ad hoc fitness function, (3) the ratio of the number of organisms "born" (i.e. as a result of mating) to the sum of the total number of organisms (both born and artificially created by the GA) and (4) the difference between food-energy gained and food-energy expended over the sum of these two values. These last two can be seen as the most important gauges of the course of the simulation. The third one for example directly indicates the progress towards an SBS, given that 0.0 indicates no naturally created creatures have been born, while 1.0 indicates that all current creatures are a product of mating behavior. The fourth one ranges from -1.0 to 1.0 and should converge to 0.0 in a world where energy is appropriately conserved. Given that one way to look at this world is as a somewhat complex energy balancing problem, this fourth analysis also allows to gain insights concerning the design choices of the virtual ecosystem; for instance how much food has to be inserted into the virtual ecosystem to allow for survival without making it too easy for the creatures.

For a detailed discussion of the results, including an analysis of the different species[8] and emergent behaviors encountered in PolyWorld, we kindly refer the reader to the original

7. Some simulation runs already acquire an SBS in the first seed population, while others never acquire it (and are thereby discarded as unsuccessful simulations).

8. "Species" here refers to a group of creatures carrying out a common individual behavior that results in distinctive group behaviors.

paper by L. Yaegar [25].

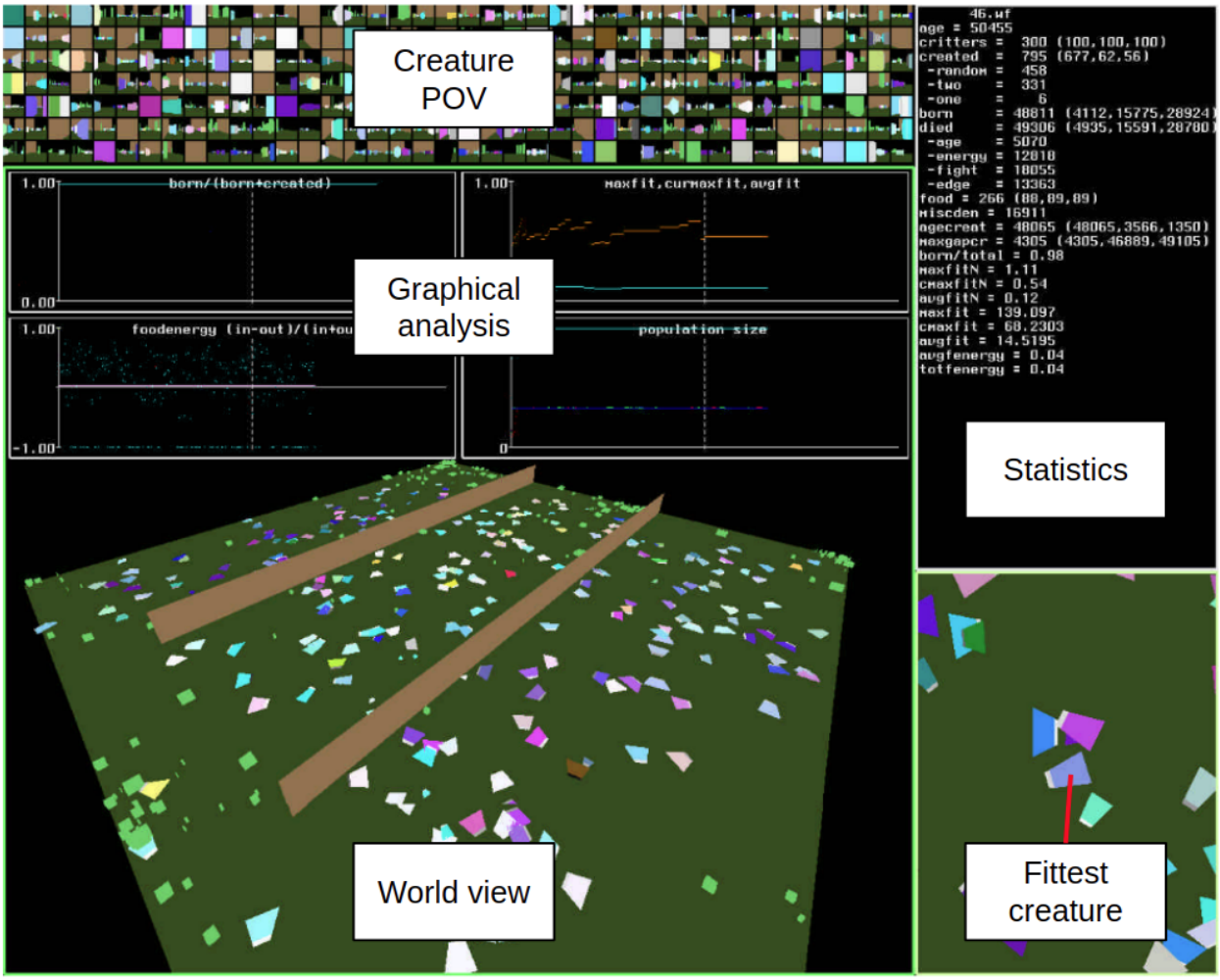


Figure 2.4: A screenshot of the PolyWorld ecological simulator, as given in the original paper [25] (boxed annotations were added for clarity). The largest panel shows a broad view of the world: the dark green ground plane, the brown impassable barriers, the bright green pieces of food and the multicolored creatures. Just above this world view are four graphs denoting various simulation parameters. At the top of the figure, the views of the world are shown from the point of view of each creature (corresponding to their visual input). At the top right, a few numerical statistics describing the simulation are shown. At the bottom right, a close-up view of the current "fittest" creature is shown.

2.3 Compositional Pattern Producing Networks (CPPN)

Before discussing the related works regarding morphological evolution and neuroevolution, this Section takes a small detour and introduces an important type of neural network used throughout these works and this thesis: *the Compositional Pattern Producing Network* (CPPN).

The CPPN was first introduced by K. Stanley in [14]. In essence, this is nothing more than a typically small neural network in which each node can have a different activation function such that the network as a whole makes a *composition* of different functions. This type of neural network is then often used to paint patterns within some n-dimensional space, by querying a value for each n-dimensional point within that space. Therefore, the CPPN receives the point's coordinates as input. The resulting shape of the produced patterns is dependent on the activation functions used. Thereby, the patterns reflect the same properties as exhibited by these activation functions. The choice of activation functions therefore allows to steer the patterns towards desirable properties such as symmetry, repetition, and repetition with variation [14, 62]. As illustrated in Figure 2.5, whereas including a Gaussian function can lead to a symmetric output pattern, a periodic function such as sine allows for segmentation through repetition. Most importantly, *repetition with variation* (e.g. such as the fingers of the human hand) is easily discovered by combining regular coordinate frames (e.g. sine and Gaussian) with irregular ones (e.g. the asymmetric x-axis) [62]. Some example patterns produced by CPPNs are shown in Figure 2.6.

The potential for CPPNs to represent patterns with motifs reminiscent of patterns in natural organisms has been demonstrated in several studies [14, 62]. Furthermore, granted that CPPNs are a superset of traditional artificial neural networks, which can approximate any function [63], CPPNs are universal function approximators as well. This means that a CPPN

can encode any pattern within its n-dimensional space. This property, combined with the desirable regularities of its patterns, makes it an attractive genetic encoding. However, to use CPPNs as a genetic encoding, the evolutionary operators (mutation, crossover) must be applicable to them. This was made possible by the *NEAT* [64] algorithm discussed in Section 2.5.2. On a final note, CPPNs belong to the class of *indirect* encodings. Whereas in a direct encoding scheme the genotype directly maps to the phenotype, an indirect encoding scheme can be seen as a blueprint or recipe from which the phenotype can be "generated".

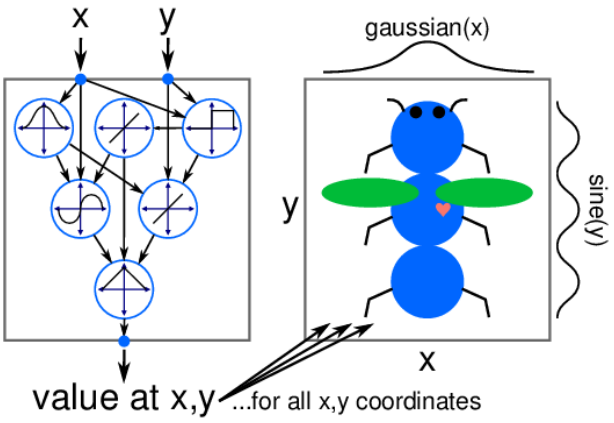
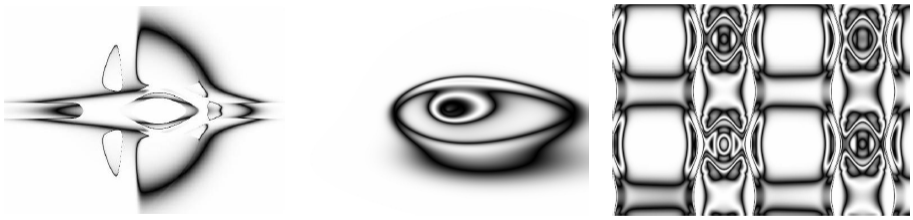


Figure 2.5: An example of a Compositional Pattern Producing Network, as shown in the paper [65]. A CPPN is typically used to query a value for each location in some geometrical space, with the property that the resulting set of values forms a pattern within that queried space. The kind of pattern that is generated depends on the underlying activation functions used. In this case, the symmetry is a byproduct of using a Gaussian activation function and the repetition of segments is a consequence of using periodical functions such as sine.

2.4 Morphological Evolution

Within the domain of evolutionary robotics, a lot of research has already been done on (simulated) morphological evolution. One of the fundamental works within this domain by K. Sims [61] evolves creatures optimized for movement in a three-dimensional environment. The creatures are represented as directed graphs (see Figure 2.7 for an example) and this graph based genetic encoding became the foundation for many other morphological experiments



(a) Symmetry (b) Imperfect Symmetry (c) Repetition with Variation

Figure 2.6: An example of some interesting regularities of the patterns produced by a Compositional Pattern Producing Network, as shown in the paper [14]. Drawing inspiration from biology, an interesting encoding scheme is that of developmental encoding. The idea is that all genes should be reusable at any point in time during the developmental process and at any location within the individual. CPPNs are an abstraction of this concept that have been shown to be able to create patterns of repeating structure in Cartesian space.

later on (e.g. Framsticks [66]). RoboGrammar [11] introduced a more recent graph-based approach for generating optimized robot structures through evolution. Within their work, Zhao et al. represent each robot design as a graph and use a graph grammar to express possible arrangements of physical robot assemblies. Doing this, each robot design can be expressed as a sequence of grammar rules. Using only a small set of rules, the grammar can already describe hundreds of thousands of possible robot designs. Furthermore, with the grammar one can constrain the design space such that the resulting designs can all be physically fabricated for a given use case. An example graph genotype with its corresponding robot morphology is given in Figure 2.8. While the morphological evolution techniques described above have focused on evolving rigid robots, research has also been carried out within the soft robotics field. The major research direction for evolutionary soft robotics steers towards using Compositional Pattern Producing Networks (CPPNs) as the underlying indirect genetic encoding. As described in Section 2.3, CPPNs have the rather unique ability of being able to paint patterns within some geometrical space. Based on the activation functions we supply them with, the produced patterns exhibit properties such as symmetry, asymmetry and repetition (with variation) of segments. As these are attractive properties for

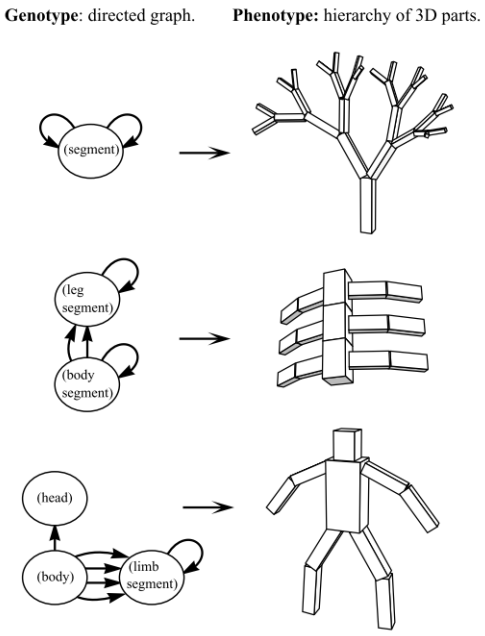


Figure 2.7: Some designed examples of the genotype graphs and corresponding creature morphologies as given in the original paper by K.Sims [61].

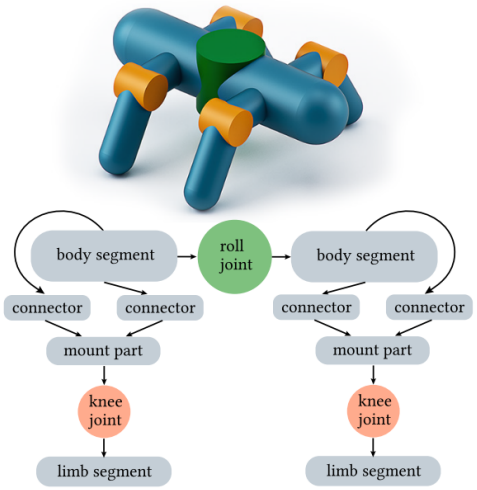


Figure 2.8: An example of the graph genotype and corresponding robot morphology of the RoboGrammar approach [11]. RoboGrammar defines a set of modular building blocks (labels of the nodes in the graph) that can be used by a graph grammar to construct the morphology graph. Note its similarity with the graph based encoding of K. Sims shown in Figure 2.7.

robot morphologies, the CPPN establishes a promising genetic encoding for evolving robot morphologies and has successfully been used within both the rigid [67, 10] and soft robotics field [15, 6]. Figure 2.9 further concretizes by illustrating how a CPPN encoding can be mapped to the resulting morphology.

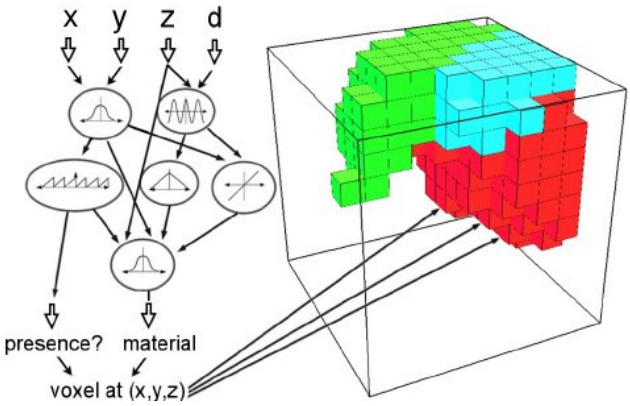


Figure 2.9: Visual example of how a CPPN encoding is mapped to a 3D (soft robot) morphology as given in [6]. In this case, the robot morphology is composed out of a set of connected building blocks with different materials. This morphology is extracted from the CPPN by querying values for specific locations (coordinate points) within a predefined box shaped space. The queried values indicate whether or not there should be a block present at the queried location (*presence*) and if so, the material that should be used for that block (*material*). In this way, the pattern of *presence* and *material* values that the CPPN generates are interpreted as a robot morphology.

2.5 Neuroevolution

Next to an optimised morphological design, an equally important part is being able to control it as well. Within the domain of robotics, this is often done through the use of Reinforcement Learning (RL) [68, 69]: an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning often comes down to training one (or multiple) underlying (deep) *Artificial Neural Networks* (ANN) through backpropagation and variants of stochastic gradient descent using many state-action-reward samples.

An alternative approach to optimising these underlying neural networks comes from the field of neuroevolution (NE), which harnesses evolutionary algorithms to design and optimize neural networks. The neuroevolution approach is inspired by the fact that natural brains themselves are the products of an evolutionary process. In line with the targeted research objectives depicted in Section 1.1, the aim of this thesis is to evolve intelligent agents and consequently neuroevolution constitutes a big part of this work.

Based on a great review by Stanley et al. given in [33], Section 2.5.1 gives a more general overview of the most important milestones and works within the neuroevolution domain. Next, the focus of the described literature is refined towards the major neuroevolution approaches used within this thesis. Section 2.5.2 describes the fundamental *Neuroevolution of Augmenting Topologies* (NEAT) algorithm. Afterwards, Sections 2.5.3 and 2.5.4 respectively describe the HyperNEAT and ES-HyperNEAT extensions of the NEAT algorithm. These both make use of the *Compositional Pattern Producing Network* discussed in Section 2.3 to indirectly encode and evolve artificial neural networks.

2.5.1 General Overview of the Neuroevolution Domain

At first, researchers focused solely on evolving only the weights of small, fixed-topology ANNs, as an alternative to backpropagation [33]. Interest quickly shifted to more ambitious possibilities such as evolving the topology (the architecture) [70, 71] and even the dynamics of intra-life learning (evolved rules for updating weights as an alternative to backpropagation-based reinforcement learning) [72, 73].

While early algorithms for evolving network topologies simply mutated weights stored in connection matrices [74], more sophisticated encodings for describing and manipulating the underlying graphs themselves also gained interest [75]. Indirect encodings, where the genome

is a recipe for generating a network rather than a direct (weight-by-weight) description of the network, also became popular [76, 77]. This allows to more compactly capture regularities such as symmetry in structure [33], an important foundation for the approach described in Section 2.5.3.

The early successes in the field often concerned evolving neural network controllers for robots, known as evolutionary robotics [78, 32, 79]. One prominent success was to produce the first running gait for the Sony Aibo robot [80]. Many notable accomplishments also exist outside of the evolutionary robotics domain, such as helping to discover the most accurate measurement yet of the mass of the top quark [81] and innovative video game concepts such as evolving new content in real time while the game is played [82]. Neuroevolution also poses a suitable test bed to study open questions in evolutionary biology, for instance the origins of the regularity, modularity and hierarchy found in biological networks like the neural networks in animal brains [83, 84].

Although impressive, especially in their day, all of these successful applications involved small neural networks by modern standards, composed only of hundreds or thousands of connections instead of the millions of connections commonly seen in modern deep neural network research [33]. This naturally raises the question whether or not evolution is up to the task of evolving such large deep neural networks. Recent studies in the neuroevolution domain [85, 86, 87] have answered this question by showing competitive performance with (and even outperforming) some of the most powerful deep reinforcement learning algorithms available today.

The core strength of neuroevolution and evolutionary algorithms in general lies in their ability to be highly parallelizable, which allows them to take great advantage of the vast

computing resources available today. These results are important because they also foreshadow the potential for neuroevolution to make an impact across the spectrum of neural network optimization problems at modern scale [33] , including cases such as (1) architecture search where differentiation is not a clear solution and (2) weight optimization without a sufficient amount of data or interaction.

2.5.2 NeuroEvolution of Augmenting Topologies (NEAT)

The NeuroEvolution of Augmenting Topologies (NEAT) algorithm was first introduced in 2002 by K. Stanley in [64]. Being the first approach to enable the application of the typical evolutionary mechanisms (mutation and crossover) to artificial neural networks in a stable manner, it can be seen as one of the fundamental genetic algorithms within the neuroevolution domain. Traditionally, the neural network topology is chosen by the human experimenter based on their prior knowledge and experience with the problem domain and neural networks in general. The effective connection weight values are then learned through a training procedure while keeping the topology fixed. This often leads to the situation wherein a trial and error process may be necessary in order to determine the appropriate topology and this is where the NEAT sets itself apart. Being an example of a topology and weight evolving artificial neural network, or TWEANN in short, NEAT attempts to simultaneously evolve weight values and an appropriate topology for a neural network. To do so, it relies on three key techniques: (1) tracking genes with historical markers to allow crossover among ever-changing topologies, (2) applying speciation (*"the evolution of species"*) to preserve innovations and (3) developing topologies incrementally from simple initial structures (*"complexification over time"*).

Genetic encoding An example of the underlying direct genetic encoding used to represent neural networks in NEAT is shown and further detailed in Figure 2.10. As it is a direct

encoding scheme, every connection and (hidden) neuron is explicitly represented. This makes the algorithm more straightforward to understand and implement, but as later discussed in Section 2.5.3 has some downsides.

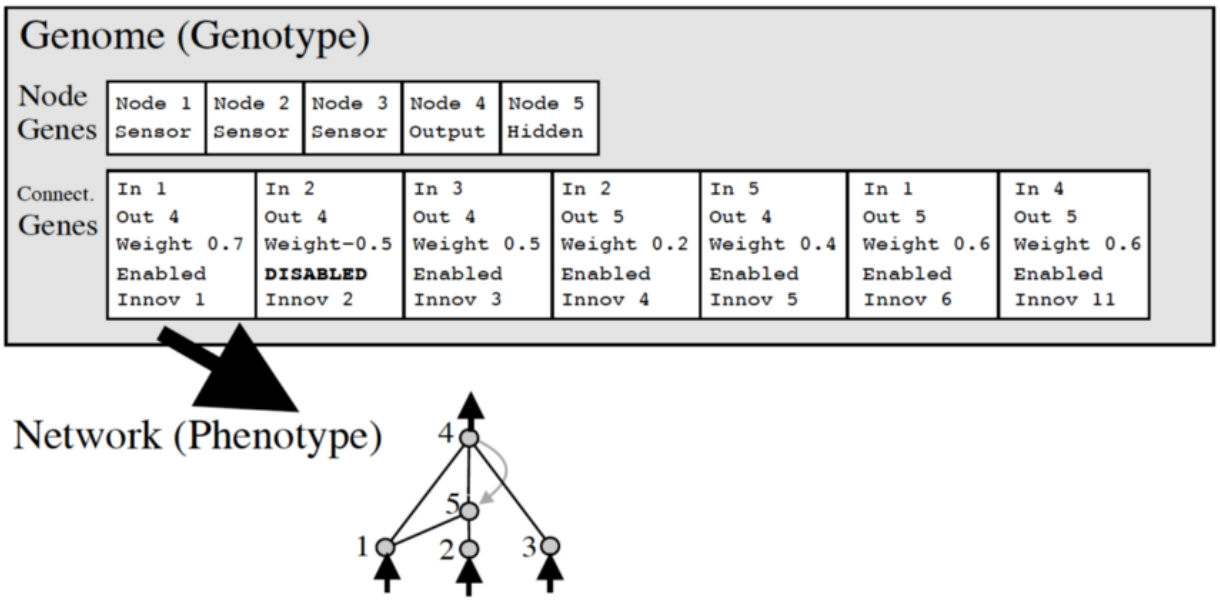


Figure 2.10: An example of the direct encoding used in the NEAT algorithm, as shown in the original paper [64]. Whereas input and output nodes in the node genes list are not evolved, hidden nodes can be added or removed through mutations. Connection genes specify for each connection where it comes into and out of, the weight of the connection and whether or not it is enabled. Additionally an *innovation number* is stored. The innovation numbers are historical markers that identify the original historical ancestor of each gene and are used to address the *Competing Conventions Problem* discussed below.

Mutations Like every other genetic algorithm (cfr. Section 2.1.1), NEAT starts off by generating a random population of these genetic encodings, which initially represent simple perceptron-like feed-forward networks of randomly connected input and output neurons. These networks are then evaluated using some score measure, allowing us to select the highest scoring genomes as the basis to create the next generation of individuals. As re-evaluating the same genomes over and over would be of little use, we introduce change in the genomes using mutation and crossover. In NEAT, mutations can either modify the weights of existing connections or alter the structure of the network by adding or removing nodes and

connections. The addition of new nodes and connections to the network is further illustrated in Figure 2.11.

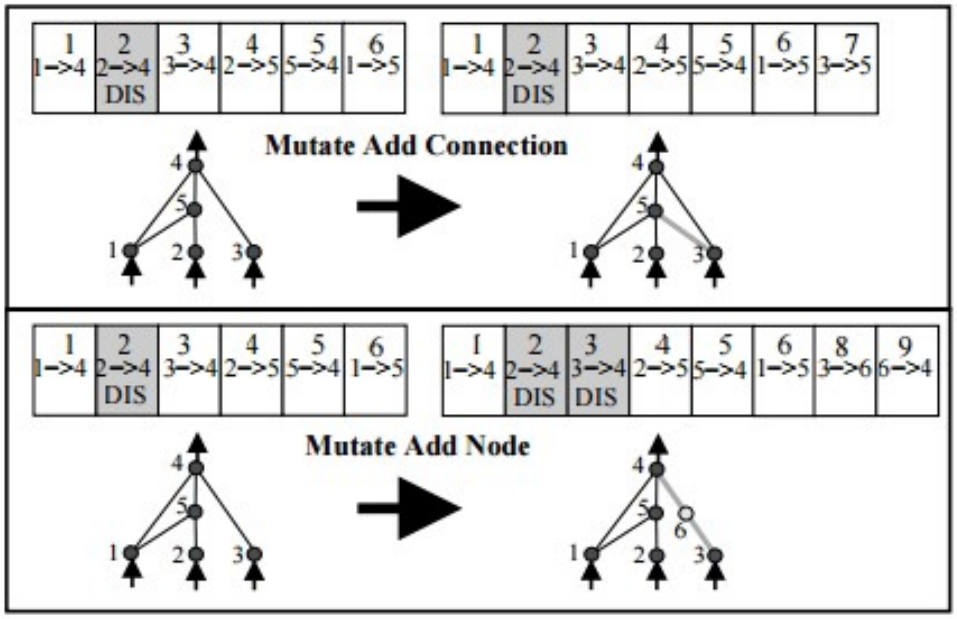


Figure 2.11: The two types of structural mutations in NEAT, as depicted in the original paper [64]. Both types, adding a connection and adding a node, are illustrated with the underlying connection genes shown above the networks. The top number in each connection gene shows its *innovation number*. If a new connection is added, it is randomly assigned a weight. If a new node is added, it is placed between two nodes that are already connected. The previous connection is disabled (though still present in the genome). The previous start node is then linked to the new node with the weight of the old connection and the new node is linked to the previous end node with a weight of 1. This was found to help mitigate issues with new structural additions.

The Competing Conventions Problem While the mutation operator of the NEAT algorithm can be seen as fairly trivial, one could consider the proposed crossover operator to be more elegant. Blindly crossing-over the genomes of two neural networks often results in networks that are damaged or rendered non-functional. If two networks for example are dependent on central nodes that both get recombined out of the network, an issue arises as shown in Figure 2.12. Within the neuroevolution domain, this issue is known as the *Competing Conventions Problem* [88, 89] and occurs when there is more than one way to represent

the neural network. Moreover, the genomes can be of different sizes. All of this raises the question *"how do we align genomes that don't seem to be naturally compatible?"*. In biology, this is supported through a process named *homology*: the alignment of chromosomes based on matching genes for a specific trait. Homology allows crossover to happen with a much lower error rate than if chromosomes were blindly mixed together.

NEAT draws inspiration from homology and tackles this *competing conventions* problem through something called historical markings. Each time a structural mutation occurs, such as adding a new node or connection, a unique *innovation number* or *historical marking* is assigned. If an identical mutation occurs in more than one genome, they are both given the same marking. Just like in homology, this allows to align genes before crossing them over and reduces the amount of erroneous configurations. A graphical example is shown in Figure 2.13.

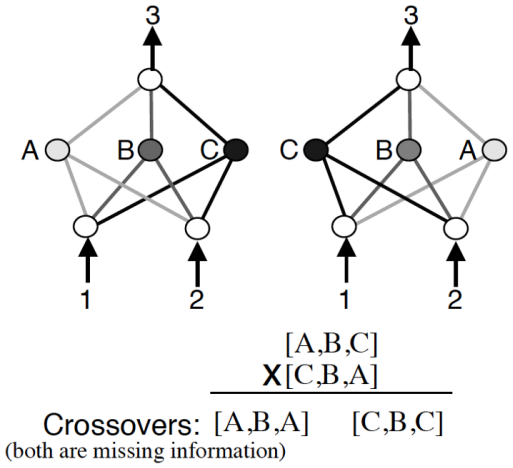


Figure 2.12: An example of the *competing conventions problem* as shown in [64]. The two networks compute the same function even though their hidden nodes appear in a different order and are represented by different genomes, making them appear incompatible for crossover. Both shown crossovers result into loss of information as important central nodes are thrown away.

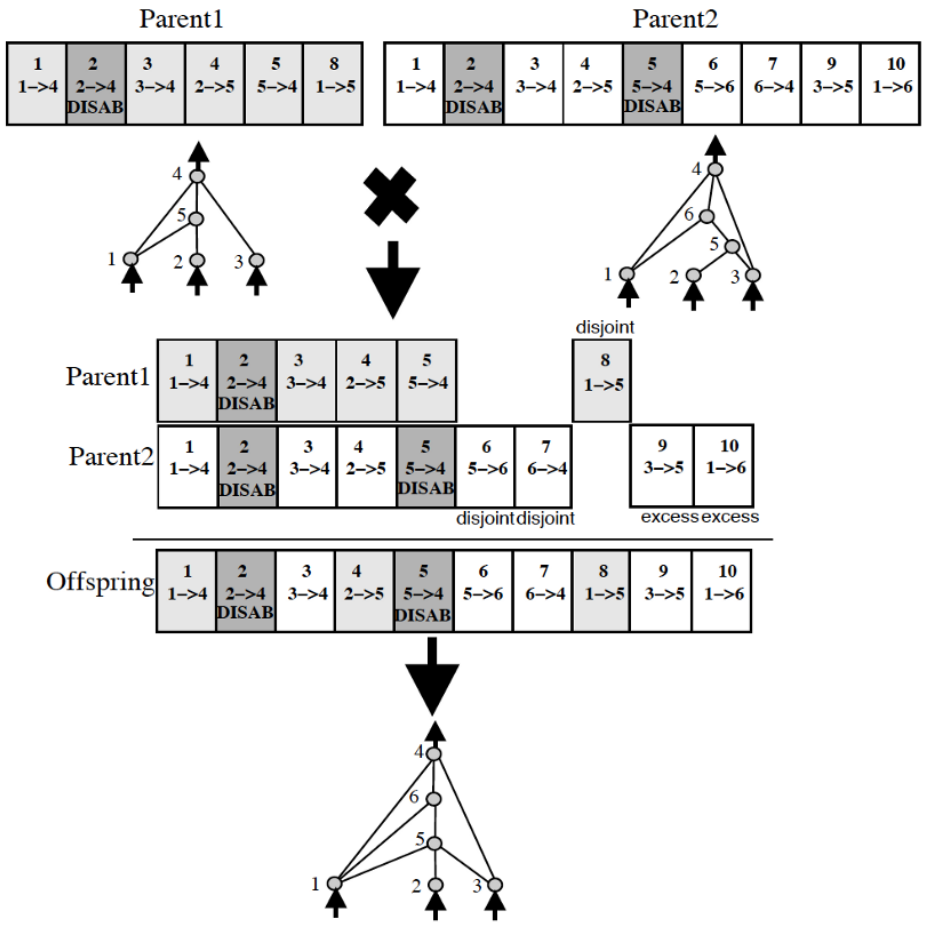


Figure 2.13: An example of crossing-over two NEAT networks, as given in the original paper [64]. The genes of parent 1 and 2 are first aligned using their historical markings. Based on that alignment, the child genome receives a functional configuration.

Innovation Protection through Evolution of Species Especially when evolving artificial neural networks, most newly evolved networks will not work well out of the blue as they require at least some weight optimization. Consequently, this puts new structures at a disadvantage compared to older structures which had some generations to further tune their weights. In order to protect new structures from being unfairly eliminated from the population, NEAT again draws inspiration from biology and employs a technique called *speciation*.

As its name suggests, *speciation* partitions the population into several species based on some similarity measure between the underlying genes of the evolved neural networks. Whereas devising such a similarity measure would have been hard if the *Competing Conventions Problem* described above still prevailed, we can simplify this by first aligning the genes using the historical markings. This alignment then allows to define such a similarity measure rather easily [64].

As the population is now partitioned into several species, new structures only compete for selection with similar and equally new structures within their own species. Consequently, these new structures are protected from unfair comparisons with older and better tuned networks. This protection mechanism is appropriately named *Innovation Protection*.

Incrementally increasing Network Complexity As mentioned above, NEAT begins with a population of simple networks that consist purely out of randomly connected input and output nodes, i.e. there are initially no hidden nodes present. As evolution progresses, structural mutations can increase the complexity of the networks by adding hidden nodes and connections. These novel structures are protected by the speciation technique described above, allowing this topological diversity to be gradually introduced over time.

This gradual production of increasingly complex structures constitutes *complexification*. In

other words, NEAT searches for the optimal network topology by incrementally complexifying existing structure. As new structure will only be kept if it is found to be useful (e.g. through fitness evaluations), NEAT has the propensity to only add complexity if and only if it is found to be necessary to increase performance.

2.5.3 HyperNEAT

Although NEAT showed promising (even state-of-the-art at the time) results within typical Reinforcement Learning problems such as the pole-balancing benchmark [90], it was severely disadvantaged to grow larger neural networks due to its direct encoding approach. Albeit intuitive, the direct encoding required every node and connection to be evolved separately. But in order to evolve a network like the human brain (with billions of neurons and trillions of connections), one would need a much faster way of evolving that structure. Moreover, biological brains and morphologies in general are known to exhibit properties such as repetition of structure (reusing a mapping of the same gene to generate the same physical structure multiple times), symmetry (mirrored structures, e.g. two eyes for input) and locality (the location of nodes in the brain influences what functionality they affect).

Indirect encoding These shortcomings of the original NEAT approach were addressed by the paper *"A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks"* [91]. Coming back to its name later, this paper proposes an extension of NEAT. The key idea is that since NEAT enables applying the evolutionary operators (mutation and cross-over) to neural networks, we can use neural networks themselves as the underlying genome for other neural networks. Instead of using NEAT to evolve the resulting neural networks directly, this extension uses it to evolve an indirect encoding capable of representing the resulting neural networks. Here the indirect encoding is the Compositional Pattern Producing Network (CPPN) described in Section Section 2.3. As its output is known to exhibit desirable

geometric properties such as symmetry, repetition of structure, asymmetry and variation, it establishes a powerful indirect encoding (see Figure 2.6). Additionally, as CPPNs can be seen as generative models, they enable phenotypic results (in this case neural networks) orders of magnitude larger than the genotypic content that represents the CPPN itself.

Reducing the search space This also introduces a shift in the search space when evolving neural networks. Conventionally in genetic algorithms, the search space is equal to the space of possible genomes (as each genome represents one candidate solution). Within the NEAT approach, this meant that the search space was equal to the space of all possible resulting neural networks (as each genome represents one distinct neural network). Within the HyperNEAT approach, the search space is reduced as it is no longer equal to all possible resulting neural networks, but only to all possible CPPNs. While the collection of all possible CPPNs (which are just a special kind of neural network) is also infinite, due to their generative characteristic described above they are often a lot smaller than the networks they represent. Therefore, the (parametric) search space in which HyperNEAT works is also smaller [91].

The HyperNEAT algorithm The main difference between NEAT and the proposed extension HyperNEAT is thus the use of a CPPN based indirect encoding which is evolved by NEAT, instead of directly evolving the neural networks using NEAT through the use of a direct encoding. This naturally raises the question of how a CPPN practically represents a neural network and how its interesting geometrical properties (symmetry etc.) are incorporated into the resulting neural network. Well, remember from Section 2.3, that the CPPN receives coordinates of points within some geometrical space as its inputs and in turn returns a value for that point. This means that in order to use a CPPN to represent or define a neural network, we first have to represent that neural network in some geometrical space.

In the original paper, the authors introduce the concept of a *substrate*. In the scope of HyperNEAT, a substrate is simply a geometrical ordering of nodes or in other words some (predefined) placement of nodes within a geometrical space. The simplest example could be a plane or grid, in which each discrete (x, y) point is a node. As depicted in Figure 2.14, a CPPN can then be used to calculate the weight of a connection between two given nodes based on their coordinates within the substrate.

This results into every node pair having some weighted connection between them (even allowing recurrent connections). Connections can be either positive or negative and often a minimum weight magnitude is defined so that any connections with weights below that threshold will result in no connection.

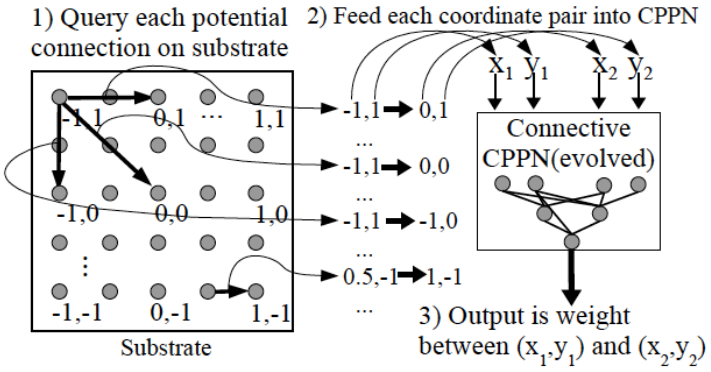


Figure 2.14: An example of how a CPPN can be used to define a neural network using the HyperNEAT approach, as shown in the original paper [91]. On the left, we see an example grid-type substrate (placement of nodes within a geometrical space). By iteratively querying the CPPN with all pair of nodes within a substrate, the values returned by the CPPN can be seen as the weight of the connection between those two nodes and thus the weight that the genome gives to that connection.

The Hypercube of Connection Weight Patterns In essence, if the nodes are arranged on a two dimensional substrate, the CPPN can be seen as the four dimensional function shown in Equation (2.2). The CPPN thus represents a four dimensional *hypercube*, in which each point represents the weight of a potential connection between two nodes. Evolving the

CPPN then comes down to evolving this hypercube of weight patterns and that is where the paper derives its name[9] from.

$$CPPN(x1, y1, x2, y2) = weight \tag{2.2}$$

Designing the substrate's shape (and placing nodes within) is done by the user of the algorithm. Because we define this substrate or "*Artificial Neural Network (ANN)* blueprint" within some geometrical space, we can seed the ANN with some geometrical information of the task it is supposed to solve. In this way, knowledge about the problem geometry can be injected into the search and HyperNEAT can exploit the regularities (e.g. adjacency or symmetry) of a problem that are invisible to traditional encodings [92]. An example of this unique property is given in Figure 2.15.

Large neural networks can be created by placing many nodes in the substrate. The connections and their weights will then be optimized as the CPPN evolves. An example of how a typical feed-forward network can be evolved using HyperNEAT is shown in Figure 2.16.

It is fairly easy to see that the desirable properties (symmetry, repetition of structure, etc.) described above will also be naturally incorporated within the evolved neural networks. These exist within the produced patterns of the underlying CPPNs and since we interpret those patterns as patterns of weights (and by extent connections), the resulting neural networks also exhibits these properties.

Substrate Resolution Scaling One final, powerful and unique property of HyperNEAT is the ability to scale the resolution (i.e. node density) of the substrate up and down. As we interpret the patterns produced by the underlying CPPN as the weights (and connections)

9. *"A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks"*

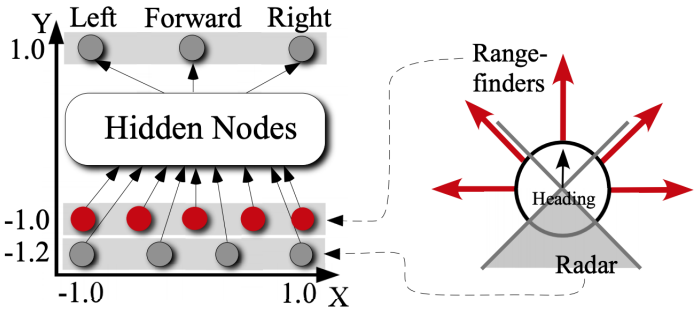


Figure 2.15: An example of making the ANN based controller aware of the geometry of the problem, as given in [92]. In this simple case, a robot configuration is shown on the right. The robot has range finder sensors ordered from left to right, which all correspond to one input node in the ANN. As the ANN is defined in a geometrical space, we can similarly order the input nodes from left to right. In the same manner, the output nodes (corresponding to going left, forward or right) receive an intuitive location in the ANN.

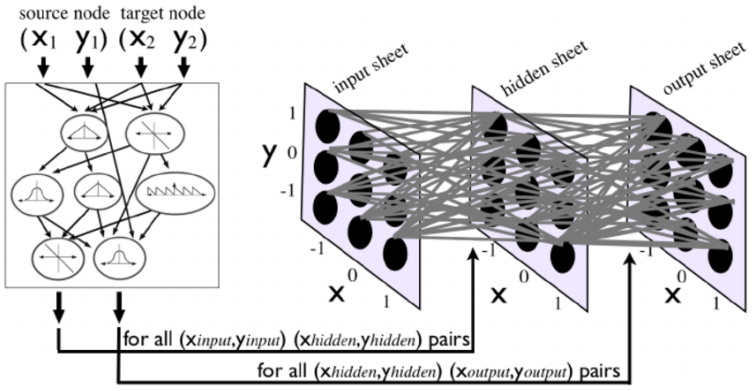


Figure 2.16: An example of how the HyperNEAT method can be used to evolve typical feed-forward networks, as given in [92]. By designing the substrate such that nodes are laid out in a layer wise manner in the geometrical space, the CPPN can be queried to connect these nodes within the subsequent layers.

of our neural network, we can easily use these same patterns (and thus the same genome) to create bigger and smaller networks with the same functionality by just requerying the CPPN at a higher or lower substrate resolution. While these higher- or lower-resolution connectivity patterns may contain artifacts that were not expressed at the resolution at which it was evolved, they will still embody a good approximation of the initial solution [91]. Experiments done in the original paper [91] show that this is indeed possible, without requiring any extra training.

2.5.4 Evolvable-Substrate HyperNEAT

Many approaches to neuroevolution mainly concentrate on deciding which node is connected to which. The HyperNEAT approach, discussed in Section 2.5.3, provided a new perspective by showing that the pattern of weights across the connectivity of an ANN can be generated as a function of its geometry. Despite its novel capabilities, a significant limitation with the original HyperNEAT is that the human user had to decide the placement and number of hidden nodes within some geometrical space called the substrate. This subsection introduces an enhanced approach that addresses this limitation and automatically evolves an optimal placement and number of hidden nodes within the substrate, starting from user-defined input and output nodes. The approach was proposed in [93] and named *Evolvable-Substrate HyperNEAT* or *ES-HyperNEAT* in short.

Searching the Hypercube for Areas of High Information The key idea in ES-HyperNEAT is to search through the hypercube of connection weight patterns painted by the CPPN to find areas[10] of high information. As each point within this hypercube stands for the weight of a connection between two nodes in the substrate (cfr. Section 2.5.3), these areas of high information can be seen as areas where the variance of those connection weights

10. As the ANN is created in a geometrical space (the substrate), it allows us to reason about it using terminology such as 'regions' and 'areas'.

is high.

The basic philosophy here is that "density should follow information": where there is more information in the CPPN-encoded pattern, there should be a higher density of nodes and connections within the substrate to capture it [93]. The major insight behind this, is that for any given pattern there is some density above which increasing density further offers no advantage. For example, if the hypercube is filled with only maximal connection weights (i.e. all weights are the same constant), then in effect it encodes a substrate that computes the same function at every node, as all nodes would be connected to all other nodes using the same weight. This means that adding more such nodes adds no new information. On the other hand, if there is some stripe of different weights running through that hypercube, but otherwise uniform maximal connections everywhere else, then that stripe contains information that would perform a different function compared to its redundantly-uniform neighbors [92]. The insight here is that this (information rich) stripe can be identified based on its higher variance.

Given the user-defined locations for the input and output nodes within the substrate, the CPPN induced hypercube of connection weights can thus be used to choose which potential connections to *express*, thereby adding the corresponding hidden nodes to the substrate along with these connections (as further explained below). Consequently by using this technique, there is no longer a need for the users to decide anything about hidden node placement or density as the underlying CPPN patterns decide this for us. Furthermore, ES-HyperNEAT can represent clusters of neurons with arbitrarily high density, even varying in density by region. This allows it to increase neural complexity in the regions where its required (e.g. the areas of the ANN where complex sensory information is processed), while keeping the complexity of the rest of the brain stable.

While the original proposal of ES-HyperNEAT brought forward the key idea of evolving the placement, density and connectivity of neurons as a whole by interpreting the patterns produced by the CPPN, its actual execution was quite expensive. This issue was mitigated in a follow up paper by the same authors [92], encapsulating the changes in a slightly modified approach named *iterated ES-HyperNEAT*. As it is just a slightly different algorithmic approach based on the same foundational idea, in literature (and throughout this thesis) it is still referred to as *ES-HyperNEAT*. What follows is a description of this iterative version of the algorithm.

The algorithm[11] consists of three main stages, which are further detailed in the paragraphs below: the *division phase*, the *pruning phase* and the *cleanup phase*. While the division and pruning phases are done at node level (i.e. for every input, output and located hidden node), the cleanup phase is done to the entire resulting neural network. Initially, there are no hidden nodes as only the user-defined input and output nodes are given. The algorithm thus starts with going through the division and pruning phases for the input and output nodes first and thereby locates a set of hidden nodes. The same mechanisms are then applied on those hidden nodes for a given number of iterations, consequently connecting them and even discovering additional hidden nodes per iteration. This also enables recurrent connections to arise. Figure 2.17 gives an overview[12].

Representing the Hypercube using a Quadtree The major task to perform is to interpret the patterns painted by the CPPN in the hypercube as a set of connections and nodes within the substrate. Thereby allowing some areas in the substrate to contain more

11. Pseudocode is available at: `http://eplex.cs.ucf.edu/ESHyperNEAT`.

12. As it is easier to visualize, the explanations will be done using a two dimensional substrate, but it is important to keep in mind that this algorithm can also be scaled to for example a three dimensional substrate (as used within this thesis).

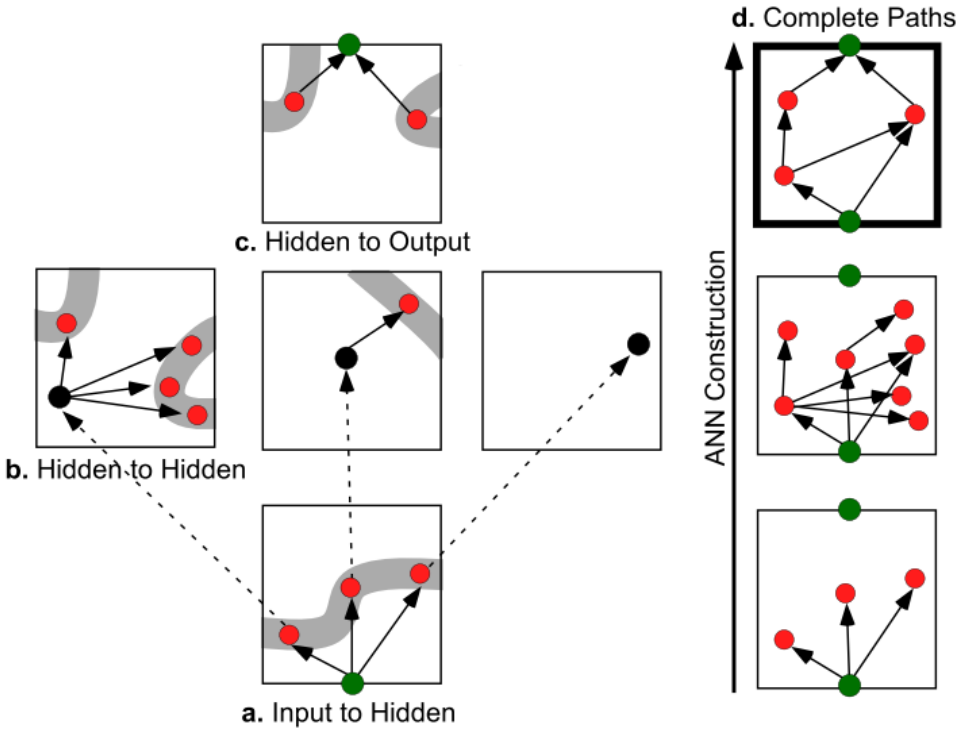


Figure 2.17: A visual overview of the (iterated) ES-HyperNEAT algorithm, as given in the original paper [92]. The algorithm begins with iteratively extracting the placement of hidden nodes, starting from the input nodes (a) and simultaneously from the output nodes (c) of the ANN. The two-dimensional motif in (a) represents the outgoing connectivity pattern from a single input node whereas the motif in (c) represents incoming connectivity patterns for a single output node. The target nodes discovered are those that reside within bands in the hypercube. The algorithm is then iteratively applied to the discovered hidden nodes (b). Only nodes that lie on a path to an input and output neuron are kept (d). That way, the search through the hypercube is restricted to functional ANN topologies.

structure based on the variance of the values in that area's hypercube location. Therefore, a spatial data structure is needed that allows that hypercube to be represented at variable levels of granularity. One such multi-resolution technique used by the authors is the quadtree [94] and allows to recursively split a (in this case) two-dimensional region into four sub-regions. That way, the decomposition of a region into four new sub-regions can be represented as a subtree whose parent represents the original region with one child node for each sub-region. To avoid confusion between ANN nodes and quadtree nodes, the ANN nodes will be called *neurons* in the following paragraphs.

Division phase Given some input, output or discovered hidden neuron, the *division phase* starts off by creating a neuron-specific quadtree. By fixing the current neuron, we only have to search for regions of high variance within the two-dimensional cross-section of the hypercube containing that neuron, instead of the full four-dimensional hypercube. This two-dimensional cross-section can be seen as the connectivity pattern that the CPPN generates for that given neuron. Every location within this two-dimensional cross-section corresponds to a neuron location within the two-dimensional substrate, for which the connectivity pattern denotes the weight of the (potential) connection between the current neuron and that neuron.

In this way, the neuron-specific quadtree only has to represent the two-dimensional slice of the hypercube containing that neuron. Each node within this neuron-specific quadtree then represents some subregion within the neuron's two-dimensional slice of the hypercube, and by extent a potential target neuron's location that the given neuron may be connected with. This target neuron location is defined as the center of the subregion that the quadtree node represents.

As its name suggests, the division phase (shown in Figure 2.18) builds this neuron-specific quadtree by recursively partitioning the neuron's two-dimensional hypercube slice until ei-

ther a minimum desired resolution is reached (user-defined minimum depth threshold) or no further subdivision seems necessary (user-defined variance threshold). The variance of a region within the two-dimensional hypercube slice represented by a quadtree node is calculated on demand by querying the CPPN weight outputs for the centers of the quadtree node's child regions. Given this variance[13] threshold, some regions might be subdivided more than others, allowing varying depths in the quadtree and consequently varying densities of potential target neurons to connect with. Additionally, a maximum resolution threshold is used to place an upper bound on the maximum depth of the quadtree, thereby limiting the number of possible neurons within the resulting neural network.

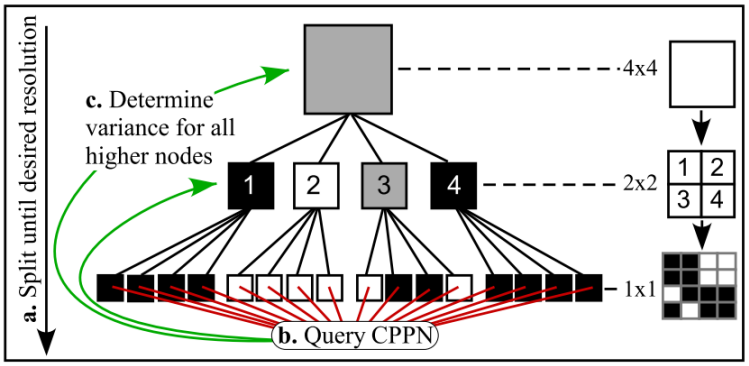


Figure 2.18: Visualization of the quadtree building in the division phase, as given in the original paper [92]. In the division phase, the quadtree is created by recursively splitting each square (a two-dimensional connectivity pattern for a given neuron) into four new squares until the desired resolution is reached (a). Subsequently, the CPPN value for each leaf (b) and the variance of each higher neuron is determined (c). The white square 2 does not meet the minimum variance threshold to decompose further. Black squares 1 and 4 (and 3 partially) meet the variance threshold and are decomposed further, consequently creating varying regional resolution.

Pruning phase The quadtree representation in the division phase serves as a heuristic variance indicator to decide on the placement and density of connections (and the target hidden neurons) to express for a given neuron. Because more structure should only be

13. Variance is used here as a heuristic indicator of the heterogeneity (i.e. presence of information) of a region.

expressed in regions with higher variance, a pruning phase is executed next as shown in Figure 2.19. In this pruning phase, quadtree nodes are removed whose parent node's variance is smaller than a user-defined threshold (different from the one used in the division phase). Here, a quadtree node's variance is calculated using the CPPN values queried for all leaf nodes within the subtree defined by that node. This phase can thus be seen as an additional variance based filter of the potential target neuron locations of the current neuron's connections.

An example of the target points chosen in the two-dimensional slice of the hypercube for a given neuron is shown in Figure 2.20 (a). As the variance is high at the borders of the circles, there is a high density of expressed points at those locations. However, the authors point out that the raw pattern output by the quadtree algorithm can be improved further. If we prune the points around borders, thereby making it easier for the CPPN to encode points safely within one region or another, we give the CPPN an explicit mechanism for affecting density. In other words, because we ignore the edges and focus on the inner region of *bands*, more or narrower bands within the patterns produced by the CPPN can be interpreted as requests for a higher point density. As you might expect, this technique is called band pruning and its result is shown in Figure 2.20 (b).

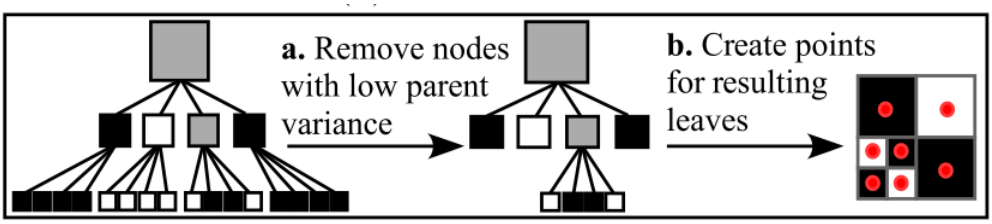
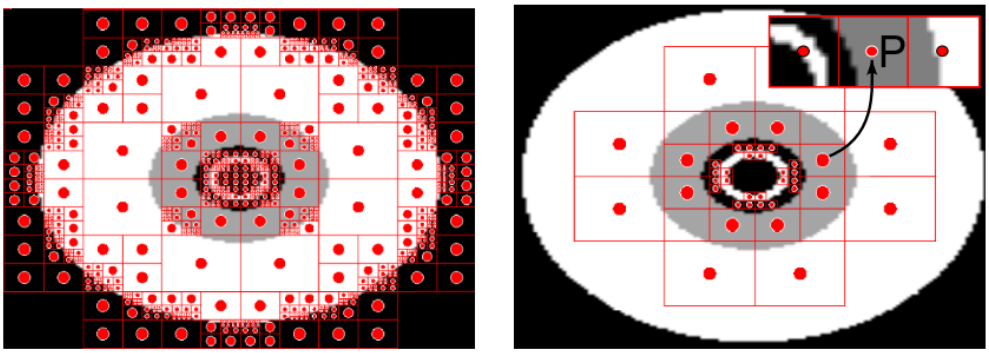


Figure 2.19: Visualization of the quadtree pruning in the pruning phase, as given in the original paper [92]. All quadtree nodes whose parent has a smaller variance than a given user-defined threshold are removed (a). The remaining quadtree leaves are then the potential locations of neurons (red points) that the current neuron can connect with (b). The density of points in different regions corresponds to the amount of information within that region.



(a) Without band pruning (b) With band pruning

Figure 2.20: Visualization of an example two-dimensional connectivity pattern produced for a given neuron by the CPPN, as shown in the original paper [92]. In (a), the red points denote the locations for potential connection targets for the given neuron that remain after the pruning phase. (b) shows those same points, after band pruning.

Cleanup phase In this final phase, the set of discovered hidden neurons and their connections are cleaned up. This comes down to removing all neurons that do not lie on a path between an input and output neuron. Whatever neurons and connections remain make up the resulting neural network. This causes the search through the hypercube to be restricted to functional ANN topologies.

Discussion In their paper, the authors show that while ES-HyperNEAT and the original HyperNEAT perform similarly on a simple maze navigation task, ES-HyperNEAT significantly outperforms HyperNEAT in a more complicated version of this task. The main reason for this is that ES-HyperNEAT has the ability to better elaborate on existing structure in the substrate during evolution, in contrast to HyperNEAT which tends to immediately start with fully-connected ANNs.

While the ES-HyperNEAT approach offers many interesting advantages such as evolving complete neural networks without requiring much human design, it is still a very experimental method and much remains to be learned about its capabilities.

Adaptive Evolvable-Substrate HyperNEAT

HyperNEAT exposed the fact that neuroevolution benefits from spatially organized ANNs, or in other words from neurons that exist at *locations* within the space of the ANN (substrate). By placing neurons at locations, evolution can exploit *topography* as well (as opposed to just topology), which makes it possible to link the geometry of sensors with the geometry of the ANN. This means that two ANNs with the same topology (i.e. the same connectivity pattern) can still differ in topography (i.e. neurons are placed at different locations in space). While lacking in many ANNs, such geometry is known to be a critical facet of natural brains that is responsible for e.g. brain modularity [95]. The biggest drawback of this approach was that the human user had to decide on the number and placement of hidden neurons in the substrate. Later, the ES-HyperNEAT extension showed that it was possible to also extract the placement and density of hidden neurons from the very same underlying indirect encoding. As the chosen indirect encoding, the Compositional Pattern Producing Network (CPPN), produces a theoretically infinite-resolution pattern of weights within the substrate, this pattern can be interpreted in such a way that neurons are automatically placed in locations that best capture the information stored within that pattern.

One remaining challenge to be addressed was the adaptiveness of the evolved networks, or in other words allowing the networks to learn during their lifetime. While some tasks do not require the network to change its behavior, many domains (especially robotics) benefit from online adaptation. While evolution offers phylogenetic adapatation, learning gives the individual the ability to react to environmental changes by altering its behavior during its lifetime. Practically, learning in neural networks comes down to changing the weights of connections in such a way that the network as a whole better operates.

Adaptive HyperNEAT One of the main approaches followed within the neuroevolution domain evolves online adaptation by encoding local learning rules in the genome. These local learning rules control how connection weights should change in response to the changing activation levels of the neurons they connect [96, 97, 98]. While promising, this approach suffers from the same issues as the NEAT approach with its direct encoding: the local learning rules for every connection in the network must be discovered seperately. HyperNEAT was able to address these similar limitations of NEAT by showing the possibility of defining the connectivity of an ANN based on patterns of weights produced in function of the ANNs geometry. This gave rise to the idea that the learning rules of an ANN could also be defined by a pattern within its geometrical space, i.e. a pattern of local neural learning rules. Comparing to biological brains, this also better reflects the intuition that DNA does not seperately encode the plasticity of every synapse in the brain.

Further building upon the CPPNs encoding power, the *adaptive HyperNEAT* extension proposed in [99], augments the CPPNs output in such a way that it can also produce patterns of local neural learning rules (next to the original patterns of weights). These learning rules take the form of the Hebbian plasticity rule proposed in [97]: $\Delta w_{ji} = \eta * [Ao_j o_i + Bo_j + Co_i + D]$, where η is the learning rate, o_j and o_i the activation levels of the presynaptic and postsynaptic neurons and A to D are the correlation term, presynaptic term, postsynaptic term and a constant respectively. The resulting structure of the CPPN is shown in Figure 2.21.

Adaptive ES-HyperNEAT As the ES-HyperNEAT extension uses the same underlying CPPN encoding as the original HyperNEAT, it can be easily further extended into *Adaptive ES-HyperNEAT* as proposed in [100] by using the same augmented CPPN. The main difference is the additional CPPN output M, which enables an additional learning mechanism: *neuromodulation*.

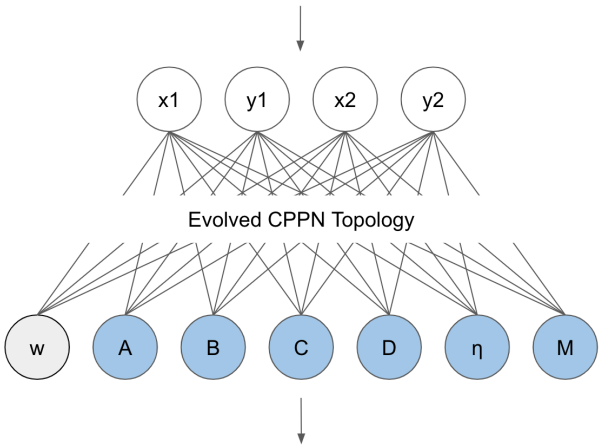


Figure 2.21: An example of the augmented version of the CPPN that allows producing patterns of local neural learning rules, as given in [100]. The CPPN is activated once for each queried connection to determine its weight (gray) and learning parameters (blue). While the output M is not used in Adaptive HyperNEAT, it is used in Adaptive ES-HyperNEAT to allow for connection mediated neuromodulation.

Neuromodulation In a neuromodulated network, the activation level of some neurons can influence the plasticity of other neurons directly. This allows the ANN to change the level of plasticity of specific neurons at specific times, a property that seems to play a critical role in regulating learning behavior in animals and has shown to benefit ANNs in more complex, dynamic, reward-based scenarios [100]. Within the geometrically organized ANNs evolved by ES-HyperNEAT, neuromodulation could thus allow one brain region (e.g. where the sensory information is processed) to directly influence the connection weight updates of another region (e.g. where the robot's movement is determined).

Practically in ES-HyperNEAT, this mechanism is implemented through *connection mediated* neuromodulation, where connections instead of neurons are either standard or neuromodulatory[14]. Before the weights of the incoming connections of a neuron are updated, the modulatory activation of that neuron is now first computed. This modulatory activation

14. Neuromodulatory connections are connections that change the plasticity of the postsynaptic neuron, based on the activation level of the presynaptic neuron.

is just the typical (connection-)weighted sum of the activation levels of all other neurons by which the neuron is connected through neuromodulatory connections. Quite similar to the static learning rate η in the Hebbian learning rule described above, the modulatory activation m_i of a neuron i then has the same effect, although in this case it can dynamically change based on other parts of the ANN: $\Delta w_{ji} = tanh(\frac{m_i}{2}) * \eta * [Ao_j o_i + Bo_j + Co_i + D]$

To decide which connections are standard and which are neuromodulatory, we use the CPPN output M shown in Figure 2.21. Depending on the activation function used, a connection can for example be set as neuromodulatory if its CPPN output has a positive M.

Part I

Evolving Virtual Creatures

CHAPTER 3

ONE CPPN TO RULE THEM ALL (OCRA)

The Compositional Pattern Producing Network (CPPN) described in Section 2.3 establishes the central entity within this thesis. The key insight behind its usage, is that the CPPN allows to produce geometrical patterns within some queried space. Depending on the underlying activation functions we allow the CPPN to evolve, these patterns exhibit properties such as symmetry, asymmetry, repetition of segments and repetition with variation [14]. As these properties are really attractive for both morphologies [14] and brains [100], the CPPN induced patterns establish a promising encoding for them. Consequently, within current literature, the CPPN has been successfully used as the underlying indirect encoding to evolve both robot morphologies (Section 2.4) and brains (Sections 2.5.3 and 2.5.4), albeit never to simultaneously do both.

This is where the proposed *One CPPN to Rule them All*, or *OCRA* in short, approach differentiates itself. As its name may suggest, it comes down to representing both creature morphology and brain with the same underlying genome, the CPPN. The key insight is that when only evolving a single genome that represents both brain and morphology, this more naturally incorporates the simultaneous evolution of the two. Consequently, this should lead to the morphology and brain being more fitted for each other, as they are now derived from the same mutating *metaspace*. This property is often referred to as *embodied intelligence* and is known to play an important role in the creation of well working embodied robots [101].

The main hypothesis behind this approach is that when evolving the CPPN, it is capable of evolving in such a way that it learns an appropriate relationship between the brain and morphology defining patterns that it produces. It therefore causes the (pattern-changing) mutations on the CPPN to result in complementary and compatible mutations for both the

morphology and brain. Consequently, this approach allows to mitigate the current issues of simultaneous brain-body evolution described by Cheney et al. in [9]. The *OCRA* approach thereby establishes the main strategy to achieve research objective one, described in Section 1.1.

Given that the CPPN architecture is specifically designed such that the morphology and brain defining patterns that it produces are tightly linked to one another, it is important to first fully understand how the creature morphology and brain are built from these patterns. Therefore, the following two Sections 3.1 and 3.2 first discuss this for both the morphology and brain respectively, while initially abstracting the CPPN as "something that paints patterns within some pre-defined space" (i.e. an entity that returns some value for some queried location). Thereafter, given the concrete idea of how the patterns are actually used, Section 3.3 describes the CPPN architecture that allows to generate these patterns.

3.1 Creature Morphology

The creation of morphologies within the OCRA approach is inspired by the work of Cheney et al. [6]. In their work, the authors describe a framework which allows to convert CPPN generated patterns into soft body robots, as discussed in Section 2.4. The main difference here is the type of building blocks we make available, which in our case results into rigid morphologies composed out of connected and undeformable building blocks.

3.1.1 Agent Space

As the underlying CPPN genome returns geometric patterns within some queried space, we first have to define such a space in which we will construct the virtual creature's morphology. Within this thesis, this space will be referred to as the *Agent Space*. While any shape is possible, the most straightforward choice is a cube centered around origin. This cube can then

be further subdivided into smaller *subcubes* until a user-defined *morphological resolution* is reached. Each of these subcubes will then potentially be used as the location for a building block.

This *morphological resolution* consequently specifies the sample rate at which the underlying CPPN patterns are queried and offers an easy way to increase or decrease the resulting morphological complexity, i.e. the number of blocks used.

Morphological Building Blocks While many possible types of building blocks can be designed by both humans and evolved agents themselves, the major types proposed here are:

- **Filler Block**: a normal block used to "fill" space, without any specific function other than to connect other blocks. This block can potentially be transformed to a joint block.
- **Brain Block**: a block that contains the brain of the virtual creature. The Brain Block represents the *Brain Space* described below. Every valid morphology will have exactly one of these.
- **Sensor Block**: a block that allows the creature to perceive its environment using some (implementation dependent) type of sensor.
- **Joint Block**: a transformed filler block that contains a movable joint. Every movable morphology will at least have one of these. The parameters describing the joint's degrees of freedom can be user-defined but in this case they are also extracted from the CPPN induced patterns.

The big advantage of working with modular morphologies composed out of a user-defined list of building blocks is that it allows to reuse this same approach in other scenarios as well, for example where other types of building blocks such as *Wheel Blocks* are available.

3.1.2 Building Creature Morphology

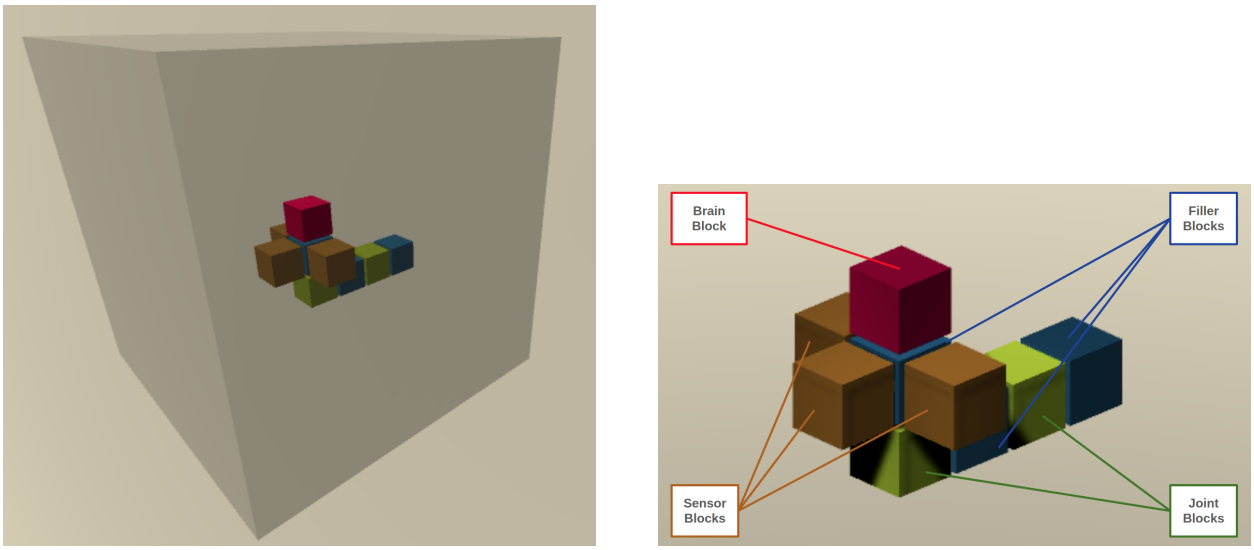
To convert the patterns produced by the CPPN into a creature morphology, we start by locating the brain block. As each creature requires exactly one brain block, this gives us an initial anchor to connect the rest of the morphology to. Originally, the idea was that this brain block could be placed anywhere within the Agent Space by allowing the CPPN to output a Brain-Block-specific pattern. Thereby the subcube location with a maximal value in that pattern would be designated as the brain block. However, as having one fixed location for the brain block allows a more stable execution of the neuroevolution algorithm used for the brain (cfr. Section 3.2), we fix the brain block at the Agent Space origin.

Given this brain block location, we then recursively query each neighbouring[1] subcube location and check whether or not there should be a building block present there and if so, which type. While respecting the user-defined resolution and size of the Agent Space, this process continues until none of the placed building blocks has a neighbouring subcube where according to the CPPN pattern a building block should be placed.

To allow the CPPN to express a specific type of building block for each subcube location in Agent Space, we let the CPPN output a specific pattern for each blocktype (cfr. Section 3.3). A block of a certain type will be placed at a subcube location, if the value of its type-specific pattern at that location is the highest amongst the different types and exceeds some activation function dependent threshold (e.g. > 0). As the brain block is fixed to origin, the remaining block types here are *filler blocks* and *sensor blocks*.

The procedure above results in an Agent Space populated with both filler and sensor blocks, centered around a brain block placed at origin. An example is shown in Figure 3.1. The only

1. Diagonal neighbours in any direction are never taken into account here.



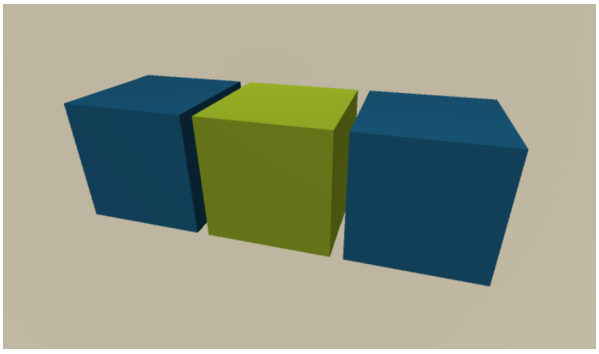
(a) A snake-like composition of different building blocks, placed within the bigger Agent Space cube.

(b) An annotated version of the snake-like composition of different building blocks.

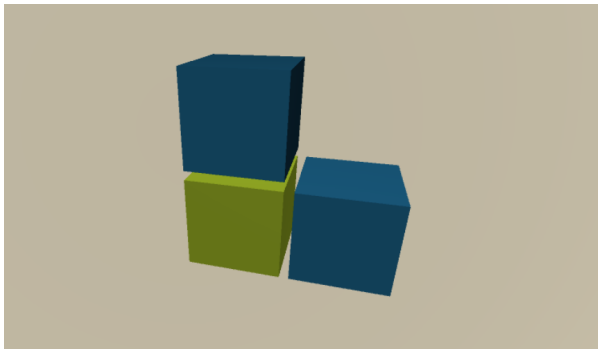
Figure 3.1: An example of the different types of building blocks placed within the Agent Space, that together form the morphology of a snake-like creature. This Agent Space is centered around a brain block (a), which is always placed at origin. (b) zooms in on this snake-like creature and indicates the different types of building blocks.

thing left to do is connecting these blocks, which in this case can be done by placing either a fixed connection or a joint based connection. Whereas fixed connections do not allow any difference in orientation between the blocks it connects, the joint based connection allows the attached block to rotate independently. This rotation of the attached block and its own (connected) neighbours forms the main mechanism behind creature movement and therefore these joints can be seen as the *actuators* of the creatures.

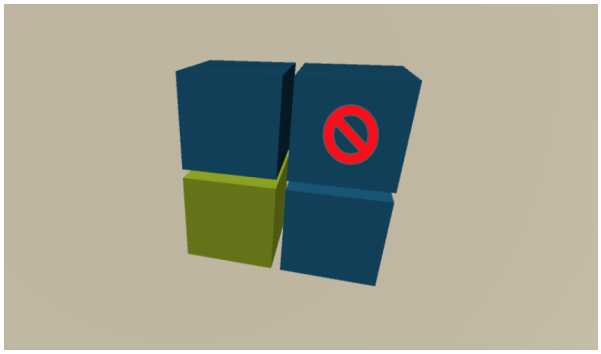
To reduce the amount of possible configurations and allow a better organization of input and output nodes for the brain (cfr. Section 3.2), we limit the joint based connections to the filler blocks. Additionally, we reduce the amount of filler blocks that are allowed to use a joint based connection by introducing the concept of a *Mount Block*. Mount blocks are a special kind of filler blocks that have two neighbours, which each have unique neighbours



(a) A valid mount block is shown in green.



(b) A valid mount block is shown in green.



(c) The potential mount block shown in green is invalid as its neighbours share a neighbour.

Figure 3.2: Three simple examples of possible mount block configurations. While (a) and (b) show valid configurations, (c) shows an invalid configuration. While the configuration in (c) could still cause significant motion depending on the configuration of the rest of the body, this will not always be the case and these are therefore filtered out for the sake of implementation simplicity.

themselves[2]. An example is shown in Figure 3.2. This allows to restrict the placement of joint based connections to blocks that can actually cause a significant motion of the morphology. For ease of implementation, we add one final restriction that allows a mount block to connect to only one of its two neighbours using a joint based connection.

To determine if a mount block connects to one of its neighbours using a joint based connection, and if so to which one of the two, we again query a CPPN produced pattern. In

2. Mount blocks are found by searching through the generated morphologies for filler blocks that conform to the specific requirements, i.e. they are not directly defined by a CPPN pattern.

this case as well, we let the CPPN output a pattern specifically meant for the purpose, i.e. joint connections. As we only allow one joint based connection per mount block, we require different output values of the CPPN pattern for each of the two neighbours to be able to differentiate between the two. Therefore, we need to feed the CPPN different input locations when querying the pattern and thus we set the queried location for each neighbour as the center of the shared face of the mount block and that neighbour. If any of the queried values exceeds some activation function dependent threshold (e.g. > 0), we attach the mount block to the corresponding neighbour using a joint based connection while using a fixed connection for the other. If they both do, we pick the neighbour with the highest value as the target for the joint based connection. If neither exceeds the threshold, the mount block is connected using normal, fixed connections to both the neighbours. If a mount block is connected using a joint connection, it is no longer a normal filler block, but a joint block. Lastly, for every joint block we then query an additional set of CPPN patterns that each describe an axis-specific degrees of freedom of the joint.

The creature morphology is now fully defined by the CPPN induced patterns, resulting in a connected composition of user-defined building blocks.

3.2 Creature Brain

After building the creature's morphology, all information required in order to build the brain using the Adaptive Evolvable-Substrate HyperNEAT (AESHN) algorithm [100] discussed in Section 2.5.4 is available: a space in which to build the brain (the brain block) and the locations of both the input (the sensor blocks) and output nodes (the joint blocks).

3.2.1 Brain Substrate

As its name suggests, the brain block is used as the space in which the brain is built and this space will be referred to as the *Brain Substrate*, in line with the terminology proposed in the original AESHN paper [100]. As stated in Section 3.1 above, this Brain Substrate is fixed to the origin in order to increase the stability of the AESHN algorithm. There are two main reasons for this: (1) AESHN employs user-defined variance threshold on the CPPN output patterns as the main mechanism behind the connection and hidden node placement (cfr. Section 2.5.4), so keeping the CPPN input values (coordinates within the Brain Substrate) within a fixed range helps to set these variance thresholds and (2) this forces all evolved CPPNs to use the same Brain Substrate, giving them some guidance towards brain-specific optimizations by changing the produced patterns there.

3.2.2 Node Placement

Given the block based creature morphology, we can easily place the *morphology dependent* input and output nodes of the brain. The morphology dependent nodes are the ones that correspond to observations made or actions carried out by the morphological building blocks. Analogous with the example given in Figure 2.15, these nodes establish a tight link between the structure of the brain and the structure of the morphology. Moreover, because both brain and morphology are derived from the same underlying CPPN, this also stimulates the CPPN towards compliant brain and morphology defining patterns, as further discussed below in Section 3.3.

Morphology dependent input nodes are located at both the sensor blocks and joint blocks. In sensor blocks, there is one input node per observation that the sensor makes. Furthermore, and this is why joint blocks also have input nodes, we place some additional input nodes that are fed observations of the current orientation and position of the block relative to the brain

block. This is inspired by the natural effect called *proprioception* or *kinesthesia* [102], which allows the body to sense its location, movements and actions based on a continuous feedback loop between sensory receptors spread throughout the body and the nervous system[3]. While it is possible that this notion of proproception does not help the virtual creatures as we expect it to do, the fact that the neuroevolution algorithm has the freedom of selecting which input nodes to connect to the brain mitigates the possibly harmful effects of these additional input nodes.

The morphology dependent output nodes are the ones that control the movable joints and are therefore placed within the joint blocks, one for each rotation axis of the corresponding joint, i.e. the X, Y and Z axis. Depending on their activation, they drive the joint in a continuous manner.

Next to these morphology dependent brain nodes, we also define *general* input and output nodes. The general input nodes allow to feed additional information to the brain, such as an oscillating sine wave to stimulate a periodic movement [19] or environment dependent observations (e.g. the amount of remaining energy of the agent in a game environment). The general output nodes allow for non-morphology dependent actions such as *"eat"* in a virtual ecosystem environment. In order to place these general nodes, we subdivide the Brain Substrate into an inner and outer region. The outer region is used for these general nodes, while the inner region remains reserved for the hidden nodes created by the AESHN algorithm. The location of the general nodes within the outer region is to be chosen by the user.

Figure 3.3 gives a schematic overview of an example snake-like creature evolved by the

3. Proproception is the reason why we for instance always have an idea of where our hands are, even if our eyes are closed.

OCRA approach. While in this case all sensor related inputs were provided using general input nodes in the outer region of the brain block, it is still a good example to showcase the link between creature brain and morphology. Furthermore, it shows that not all input nodes were connected to the brain by the AESHN algorithm, demonstrating its *feature selection* capability. The figure also shows that the input and output nodes of each joint block are placed quite closely near each other. The idea behind this is that it stimulates *brain modularity*[4] as the AESHN algorithm will connect input and output nodes that lie close together in space, to hidden nodes close together in the brain.

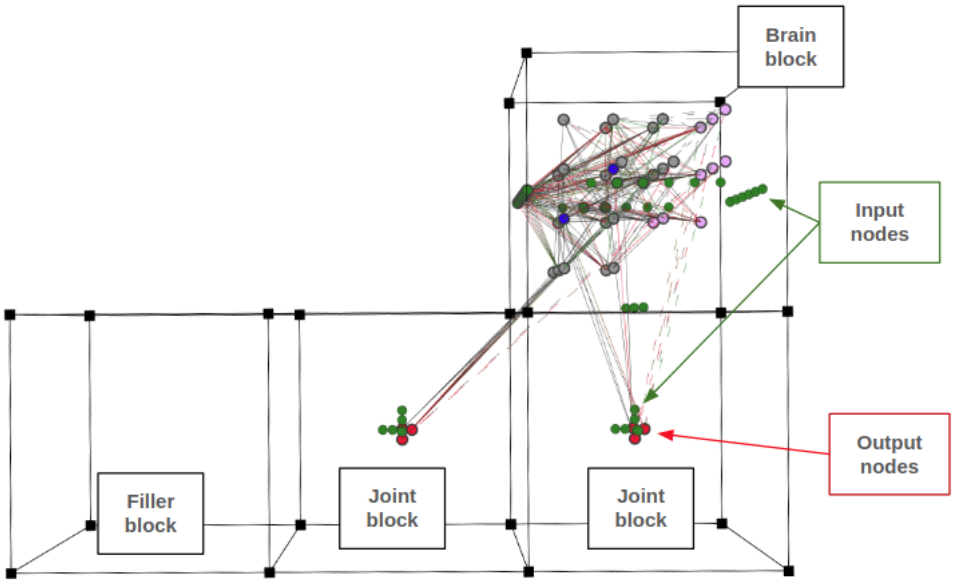


Figure 3.3: A schematic overview of a snake-like creature evolved by the OCRA approach. Whereas a sensor block could have been made available to the evolutionary process, here all sensor related input nodes are fixed to the outer region of the brain block. Every joint block contains (green) input nodes, next to its (red) output nodes. By placing these nodes quite close to each other, we can stimulate *brain modularity* as the AESHN algorithm will connect these nodes to close together hidden nodes within the Brain Substrate.

4. The theory of brain modularity suggests that there are functionally specialized regions in the brain that are domain specific for different cognitive processes [103].

3.2.3 Modifications to the AESHN Algorithm

While the main ideas behind the employed Adaptive Evolvable-Substrate HyperNEAT (AESHN) algorithm [100], described in Section 2.5.4, remained untouched, two modifications were introduced.

The first modification increases the brain-building speed by reorganizing the order in which connections (and hidden nodes) are discovered. In the original implementation, we first discover hidden nodes and connections by starting from the input nodes. Thereafter, for a user-defined number of iterations, we do the same starting from those discovered hidden nodes and possibly discover an additional set of hidden nodes. Next, we run this process again starting from the output nodes, which connects these output nodes to (a subset of) the discovered hidden nodes. Finally, we run through a cleanup stage that removes all nodes and connections that do not lie directly on a path from an input to output node.

However, if we limit the number of iterations in which hidden nodes are connected to other hidden nodes to 1, we can reorganize this order. By first discovering all potential hidden nodes, starting from the input nodes and thereafter directly from the output nodes, we can reduce the complexity of the hidden to hidden node connection discovery. Given this complete set of hidden nodes, we can skip the "hidden node discovery" stages that normally occur when we try to connect hidden nodes as we can directly try to connect these hidden nodes using only a subset of the operations normally required.

The second modification concerns itself with extending the *neuromodulation* property of the AESHN algorithm, described in Section 2.5.4. In a neuromodulated network, the activation level of some neurons can influence the plasticity (or learning) of other neurons directly. Practically in the AESHN algorithm, this was implemented through *connection mediated*

neuromodulation, where connections instead of neurons are either standard or neuromodulatory.

However, by introducing a slight modification to the cleanup phase of the algorithm, we can allow for neuromodulatory neurons as well. This cleanup phase filters out unfunctional ANN topologies by removing all hidden nodes that do not lie on a path between an input and output node. In the original implementation, this path is defined as a path made from only standard connections, i.e. neuromodulatory connections do not count. Therefore, it removes nodes which only have neuromodulatory output connections and thus by definition cannot lie on such a path. However, if these nodes are connected to the input nodes (either directly or indirectly through other hidden nodes), they can be seen as neuromodulatory nodes and should be kept.

By taking such neurons into account and avoiding their removal during the cleanup phase, we can now allow the evolution of both neuromodulatory connections and neuromodulatory neurons, i.e. neurons that can directly influence the learning of other neurons based on their own activation.

3.3 CPPN Architecture

The Compositional Pattern Producing Network (CPPN) is the indirect encoding that represents both the creature morphology and brain through the pattern that it generates. As the CPPN is evolved using the NEAT algorithm described in Section 2.5.2, we only have to choose the number (and meaning) of input and output nodes, next to the set of available activation functions that the evolved hidden nodes can apply. Whereas (most of the) the input nodes correspond to a queried coordinate location within either the Agent Space or Brain Substrate, the output nodes each produce a specific pattern within that space.

In essence, the OCRA approach combines the CPPN based morphological evolution proposed by Cheney et al. in [6] with the CPPN based neuroevolution algorithm (AESHN) proposed by S. Risi and K. O. Stanley in [100]. Therefore, we can combine their proposed CPPN architectures into one that allows to simultaneously produce both morphology and brain defining patterns. This results into the following set of (grouped) input and output nodes:

Input nodes:

- **coord1**: a group of three neurons that are fed the (x, y, z) coordinates of a brain connection's starting node.
- **coord2**: a group of three neurons that are fed the (x, y, z) coordinates of a brain connection's end node.
- **cl** (**c**onnection **l**ength): the euclidean distance between *coord1* and *coord2*. This has been shown to give the resulting brain patterns a general bias towards symmetry and to allow geometrically scaling the Brain Substrate better [104].
- **coordB**: a group of three neurons that are fed the (x, y, z) coordinates of a subcube within Agent Space.
- **bo** (**b**lock **o**ffset): the euclidean distance between the origin and *coordB*. This is meant to stimulate the morphological patterns towards symmetry about the origin.
- **bias**: a constant bias value. Here, it is set to the size of the morphological subcubes. Feeding the CPPN such a morphology dependent property has been shown to help with geometrically scaling the subcubes within Agent Space, and by extent the Brain Substrate [105].

Output nodes and their resulting patterns:

- **w** (connection **w**eight): the connection weight pattern.
- **LP** (**L**earning **P**arameters): the group of patterns that specify the connection specific learning rule parameters.
- **BB** (**B**uilding **B**locks): the group of patterns that denote the *presence* of each building block type.
- **JC** (**J**oint **C**onnection): the pattern that denotes the presence of *joint connections*.
- **JCP** (**J**oint **C**onnection **P**arameters): the group of patterns that specify joint specific degrees of freedom.

The resulting ensemble of input and output nodes is shown in Figure 3.4.

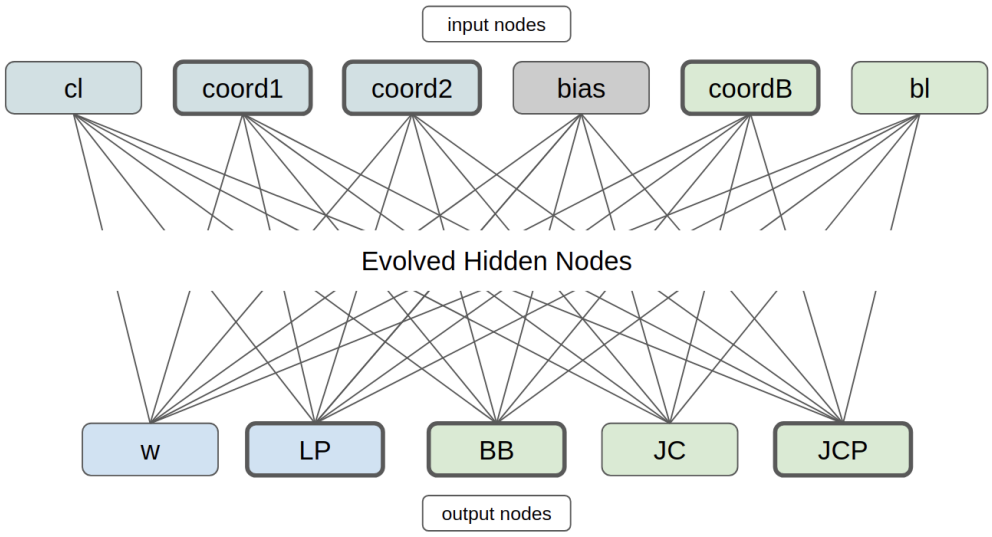


Figure 3.4: A simplified schematic overview of the CPPN architecture. Whereas the blue nodes are used for brain specific patterns, the green nodes are used for morphology specific patterns. The bigger borders around some nodes denote that they represent a group of other nodes not shown here for simplicity. For instance the **JCP** output node represents 4 underlying nodes that output **J**oint **C**onnection **P**arameters.

3.3.1 CPPN Querying

A graphical overview of all CPPN queries used to create complete embodied agents is shown in Figure 3.5. These can be seen as query templates that each allow a specific (group of) agent defining pattern(s) to be generated. The main idea is that the structure of these queries stimulate the CPPN towards evolving an appropriate relationship between both the brain and morphology defining patterns that it generates. Such a relationship can be seen as the outcome of undergoing mutations[5] that result into complementary changes to both of these patterns. In other words, we want to stimulate the mutations to occur on shared connection paths between those used for morphology queries (Figure 3.5a) and those used for morphology dependent brain queries (Figures 3.5b and 3.5d). As mutations on those shared paths will result into changes to both brain and morphology, the mutated CPPNs that best provide such complementary changes will result into better embodied agents. Consequently, the underlying evolutionary optimization process will select those more and thereby encourage the evolution of such CPPNs. Figure 3.6 gives a simple example of how mutations can resolve into changes to both morphology and brain using the proposed query strategy.

While the *coordB* group of input nodes, that receive subcube coordinates in Agent Space, could also have been left out by just using for example the *coord1* input nodes instead, it allows for some *separation of concerns*. In other words, it still allows for morphology or brain specific mutations while keeping the other untouched. An example of this is given in Figure 3.7. The other key insights behind the usage of *coordB* can be summarized as follows:

1. Using only the *coordB* Agent Space coordinate for building block and joint presence: enforces the CPPN to learn an appropriate relationship between building block and joint configurations.

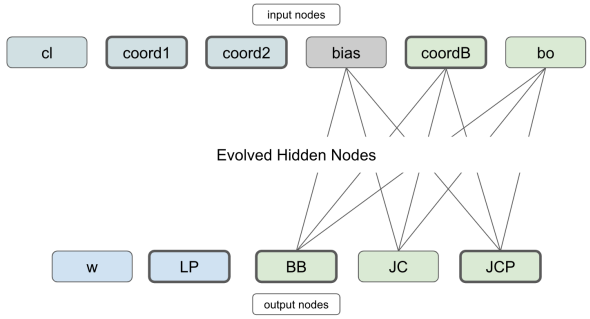
5. Given that the mutations defined in the NEAT algorithm are used here, these mutations can for example change the weight of a connection, change the activation function of a hidden node or modify the CPPN's structure by either adding or removing a node or connection.

2. Using the *coordB* coordinates for sensor and joint block inputs and outputs: enforces the CPPN to learn the relationship between the morphology configuration (blocks and joints) and the brain. This also allows the underlying neuroevolution algorithm to do morphology specific feature selection.

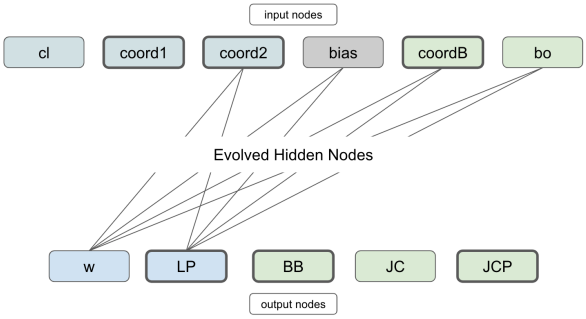
3. Setting the *coordB* coordinates to zero for general (non-morphology specific) inputs, hidden to hidden and hidden to general output brain connections: as the *coordB* coordinates cannot influence the intra-brain connections, this enforces the CPPN to learn appropriate, morphology independent and generic controls. For instance in a virtual ecosystem environment, when an agent's energy is low, the next series of actions should always lead to searching food and eating, independent of what kind of body encapsulates the brain.

3.3.2 Activation functions

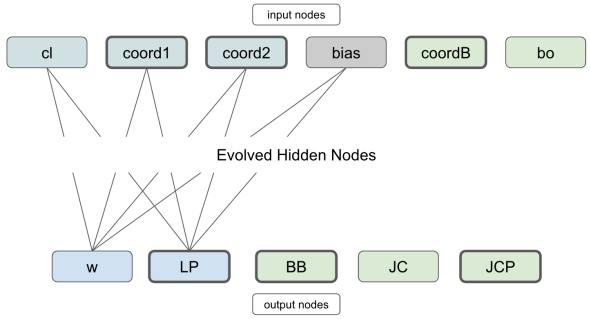
Based on some of the CPPN based works in literature [14, 100, 6, 106], the available CPPN activation functions for hidden nodes were always *sigmoid*, *Gaussian*, *absolute value*, *hyperbolic tangent*, *sine*, *identity* and the *Rectified Linear Unit (ReLU)*. The underlying formulas and a graphical overview of these activation functions is given in Appendix A. The output nodes always use the *Bipolar Steepened Sigmoid* activation function [91] (also shown in Appendix A), which causes all resulting pattern values to lie in the range of -1 to 1. This allows for instance to let the *connection* ***w****eight* pattern to produce negative weights. Additionally, this sets the threshold value to > 0 when querying the presence of a block type for instance.



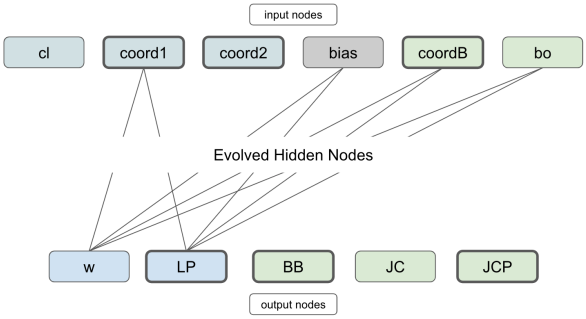
(a) Morphology queries; used to choose the block type for a given subcube location within the Agent Space and to potentially add joint connections with specific parameters.



(b) Brain queries; used to connect joint and sensor block input nodes within Agent Space to hidden nodes in the Brain Substrate.

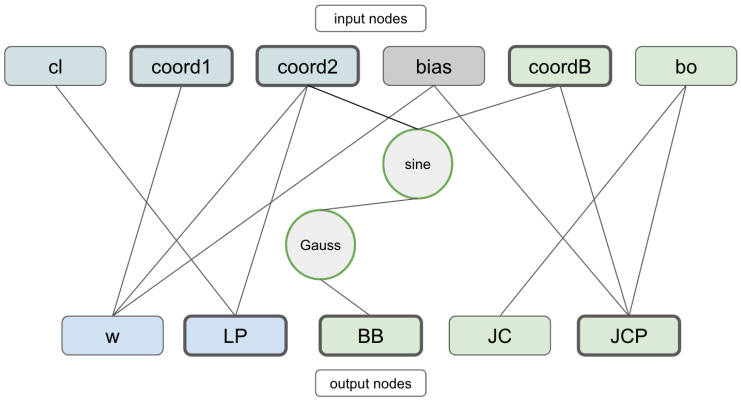


(c) Brain queries; used to connect general input and output nodes, along with hidden nodes to other hidden nodes in the Brain Substrate.

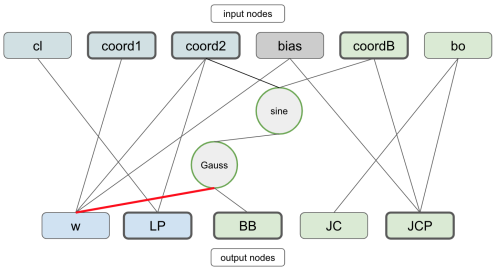


(d) Brain queries; used to connect hidden nodes within the Brain Substrate to joint block output nodes within Agent Space.

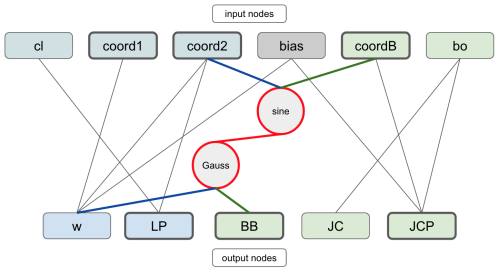
Figure 3.5: A graphical overview of the different CPPN queries used. In each query, the unconnected input nodes are set to 0.



(a) A simplified example CPPN architecture with two initial hidden nodes.

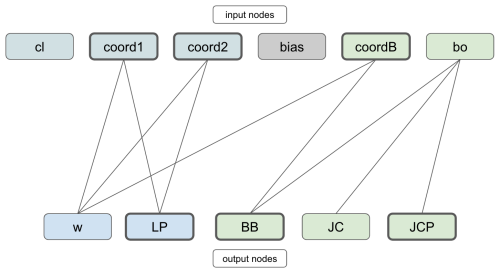


(b) A mutation occurs and a connection (red) is added between a hidden node and the *connection* ***w****eight* pattern output node.

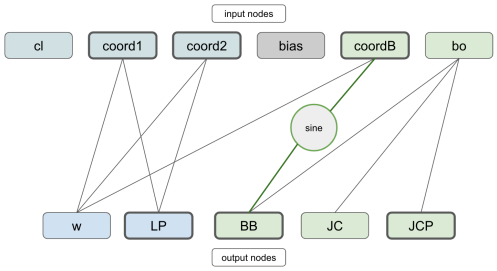


(c) The mutation introduces a shared path (red) between the the morphological queries (green) and morphology dependent input node queries (blue).

Figure 3.6: A simple CPPN architecture with two initial hidden nodes (a). Given the query strategies shown in Figures 3.5a and 3.5b, there is currently no dependence between the produced brain (w and LP) and morphology (BB, JC and JCP) defining patterns. However, a dependence arises after a mutation adds a connection (b) that results into a shared path (c) between the connections used for morphological queries and morphology dependent input node queries. Any future mutation on this shared path will therefore now introduce changes to both morphology and brain. The underlying evolutionary optimization process then guides these mutations towards compliant changes in both the morphology and brain defining patterns, by selecting the best performing ones.



(a) A simple example CPPN architecture.



(b) A mutation introduces a hidden node between one of the *coordB* input nodes and the *BB* output pattern.

Figure 3.7: An example of how the *coordB* input nodes can introduce morphology specific mutations.

CHAPTER 4

EXTENDING THE MAP-ELITES ALGORITHM

This chapter proposes an extension to the (CVT-)MAP-Elites algorithm that addresses some issues concerning its applicability in cases where the underlying genome is a neural network such as the CPPN in our case.

MAP-Elites (cfr. Figure 2.1) is a popular example of a Quality-Diversity algorithm and allows to build a ***M**ulti-dimensional **A**rchive of **P**henotypic Elites*. In other words, it builds an archive of the best performing solutions (elites), where the archive is a discretization of the behavioral (phenotypic) space instead of the parameter (genotypic) space. Here, each archive cell thus constitutes a behavioral niche. As evolution progresses, this archive is iteratively updated by selecting a number of elites from the archive, mutating their genomes and thereafter re-evaluating them in the hope that they either fill an empty archive cell or improve an existing one. Later, the CVT-MAP-Elites extension allowed to scale this algorithm to high-dimensional behavioral spaces by no longer requiring a discretization per dimension (which led to the archive size increasing exponentially with an increasing number of dimensions). Instead, it uses a ***C**entroidal **V**oronoi **T**esellation* (CVT) to divide the behavioral space into a fixed (user-defined) number of regions.

While the (CVT-)MAP-Elites algorithm has seen many successful applications [12, 29, 41, 107], it suffers from an important problem when the underlying genetic encoding is a neural network (such as the CPPN in our case). Additionally, there is a more general inefficiency concerning the reuse of available optimization information. These problems are respectively described in the two paragraphs below: *a lack of innovation protection* and *a lack of information reuse*.

A lack of innovation protection As neural networks are quite sensitive to perturbations, structural mutations (such as the addition or removal of connections and hidden nodes) often render the neural network non-functional. This leads to a disadvantage for freshly (structurally) mutated neural networks when comparing them to older ones that had more time to tune their weights. However, these structural mutations are required in order to increase the complexity of the evolved neural networks.

The NEAT algorithm (cfr. Section 2.5.2) addressed this issue by introducing the concept of *innovation protection*, i.e. protecting new structures from being discarded due to unfair comparison with older structures. Without such an innovation protection mechanism, it is highly unlikely that the evolutionary optimization technique will ever evolve more complex neural networks. Consequently, this makes innovation protection an important concept when evolving them.

This brings us to the first problem concerning the usage of MAP-Elites when the underlying genotypes are neural networks: there is no innovation protection mechanism put in place. Because MAP-Elites works by selecting some genomes stored in the archive, mutating them and thereafter directly comparing them to the current set of elites, the ones that are structurally mutated are severely disadvantaged. As expected and as shown later in the experiments (cfr. Section 5.3.3), this causes the underlying genotypic complexities to stay rather low because newer structures are unfairly compared to already tuned older structures.

A lack of information reuse In essence, the (CVT-)MAP-Elites is a genetic algorithm as it represents its candidate solutions using genomes. In genetic algorithms, the genomes are mutated in order to introduce change in the produced candidate solutions. Given the case that the genomes are neural networks (in our case CPPNs), we typically employ the mutation operators defined in the NEAT algorithm. These can be divided into two types:

(1) *structural* or *nonnumerical mutations* such as connection and hidden node addition or removal and (2) *non-structural* or *numerical mutations* such as connection weight, node bias and node response updates. Within the NEAT algorithm (and genetic algorithms in general), the mutations are often done in a random manner. In other words, the structural mutations randomly add or remove structure and the non-structural mutations randomly perturb the underlying numeric parameters by adding some value sampled from e.g. the Gaussian distribution.

While these random mutations do carry some benefits such as implementation simplicity, the major drawback is that there is no information reuse between the evaluation results of the different genomes, i.e. nothing is "learned" when a given set of parameters performs well (but not well enough to be placed in the archive) and a different set performs poorly. Additionally, it is possible that the same genome is rediscovered over and over again, causing the algorithm to waste computation by circling back to previously seen (but already forgotten) solutions.

4.1 CMA-ES infused CVT-MAP-Elites

The key insight behind the proposed extension for (CVT-)MAP-Elites is that we can mitigate the issues described above by incorporating an Evolution Strategy (ES) into the algorithm (cfr. Section 2.1.4). To do this, we have to group the underlying CPPN genomes based on their structure. A CPPN genome's structure is completely defined by the nonnumerical part of its architecture, i.e. the hidden nodes, their activation functions and how all nodes are connected. Given some CPPN structure, all what remains to be optimized are then the numerical parameters (connection weight, node bias and node response), which is exactly what the incorporated ES will be used for. As Evolution Strategies are only defined on purely numerical parameter optimization problems, this structural grouping is necessary in order to

be able to apply them. The Evolution Strategy chosen here is CMA-ES[1] (cfr. Section 2.1.4).

The general idea is to store one CMA-ES instance per genome structure in an additional archive: the *ES-Archive*. Every time a new genome structure is evolved, we create a new CMA-ES instance and add it to the ES-Archive. The major difference with the original MAP-Elites algorithm is then that instead of selecting genomes, randomly mutating them structurally and non-structurally and thereafter evaluating them, we now only randomly mutate them structurally and leave the non-structural optimization for their corresponding CMA-ES instances. Each time some genome structure would be evaluated in the original approach, we now instead continue its corresponding CMA-ES instance for a user-defined amount[2] of additional iterations; thereby optimizing that genome structure's numerical parameters in a more guided manner in contrast to the original random perturbations.

As the ES-archive enables reusing optimization information on a genome structure level (the same CMA-ES instance per genome structure is further optimized), it allows to mitigate the previously discussed issue concerning *lack of information reuse*. Additionally, an innovation protection mechanism can now also be naturally incorporated by giving new structures (and thus new CMA-ES instances) a higher number of initial CMA-ES iterations, along with a higher initial population size. This is further illustrated through pseudocode, given in Algorithm 3.

1. While we are aware that the canonical CMA-ES algorithm has issues with scaling to large dimensions [108, 51], the fact that CPPNs are indirect encodings allows them to remain a lot smaller than the phenotypes they represent. This mitigates this potential issue. Although, more appropriate ES algorithms might exist, this investigation is left for future work.

2. A user-defined amount of additional iterations allows the user of the algorithm to make the trade-off between the depth and breadth of the search. While the order of magnitude of these settings heavily depend on the computational architecture used and the complexity of the problem, as a rule of thumb within a multi-processing setup, this can be set to some fraction of the number of workers available.

Lastly, it's also possible to use multiple CMA-ES instances per genome structure instead of one. Given that the genome here is a CPPN, it is trivial to see that one such CPPN structure can generate many different patterns (and thus candidate solutions) based on its underlying numerical parameters. However, the goal of CMA-ES is to converge, which means that we somewhat "limit" the potential of one such structure. This can be mitigated by using multiple CMA-ES instances per genome structure that each work within a different area of the search space.

Algorithm 3 CMA-ES infused CVT-MAP-Elites

Parameters: G is the initial population size, I is the maximum amount of global iterations, D is the additional amount of iterations by which a selected CMA-ES instance is progressed, P is the population size used per CMA-ES instance iteration and F is the innovation protection factor.

procedure CMA-ES-CVT-MAP-ELITES(G, D, P, F)
    $A \leftarrow$ CreateElitesArchive()
    $E \leftarrow$ CreateESArchive()
    for $i = 1...G$ **do** ▷ Initialization: G random Θ (CPPNs)
        $\Theta \leftarrow$ RandomSolution()
        AddToArchive(Θ, A, E)
    for $i = 1...I$ **do** ▷ Main loop, I iterations
        $\Theta \leftarrow$ Selection(A)
        $\Theta' \leftarrow$ StructuralVariation(Θ)
        AddToArchive(Θ', A, E))
    return A
procedure ADDTOARCHIVE(Θ, A, E)
    $s \leftarrow$ GetStructure(Θ)
    $(cma, new) \leftarrow$ GetOrCreateCMAInstance(E, s)
    if new **then** ▷ Innovation protection
        $d \leftarrow F \times D$
        $p \leftarrow F \times P$
    else
        $d \leftarrow D$
        $p \leftarrow P$
    for $j = 1...d$ **do** ▷ Progress CMA instance d iterations.
        for $\eta \in cma$.sampleSolutions(p) **do** ▷ Sample non-structural parameters η.
            $\theta \leftarrow$ ReplaceNumericalParameters(Θ, η)
            $(q, b) \leftarrow$ Evaluate(θ) ▷ q is quality, b is the behavior descriptor
            cma.update(η, q)
            $c \leftarrow$ GetArchiveCell(b)
            if $A(c) == null$ or $A(c).q < q$ **then**
                $A(c) \leftarrow q, \theta$

CHAPTER 5

EXPERIMENTS

This chapter presents the experiments carried out regarding the evolution of virtual creatures using the proposed OCRA methodology. The first series of experiments, discussed in Section 5.1, concern a preliminary validation of the employed neuroevolution algorithm. Therefore, we apply it to two simple tasks, popularly used within the Reinforcement Learning domain. Given the validation of the neuroevolution algorithm, the experiments presented in Section 5.2 incorporate the morphological evolution and thereby applies the complete OCRA methodology to evolve *moving* creatures. Lastly, Section 5.3 raises complexity by incorporating sensors. Therein, the aim is to evolve *seeing* creatures, capable of moving towards targets identified by their own sensors.

5.1 Validating the Neuroevolution Algorithm

The first series of experiments concerns itself with validating the neuroevolution algorithm. In our case, this was chosen to be the Adaptive Evolvable-Substrate HyperNEAT (AESHN) algorithm introduced in Section 2.5.4, including the minor modifications proposed in Section 3.2.3. To do so, the algorithm was applied to two simple Reinforcement Learning tasks: *CartPole* (Section 5.1.1) and *MountainCar* (Section 5.1.2).

Given that these experiments were only used to validate the AESHN algorithm, we will not go in too much detail concerning their results. However, some interesting properties of the evolved controllers are showcased.

5.1.1 Cartpole

The Environment

Introduced in [109], the *CartPole* environment contains a cart which moves along a frictionless track. A pole is attached by an un-actuated joint to this cart. While the pole starts upright, the goal is to prevent it from falling over by moving the cart along its track. This is shown in Figure 5.1. Performance is measured by counting the number of time steps that the controller was able to keep the pole from falling over.

Every time step, the controller is given four input observations: (1) position of the cart, (2) velocity of the cart, (3) angle of the pole and (4) the angular velocity of the pole. Given these observations, the controller has to output an action which corresponds to either moving the car to the left or moving it to the right. Movement control is done in a discrete manner, i.e. by applying a force of -1 or $+1$ to the cart which respectively make the cart go left and right.

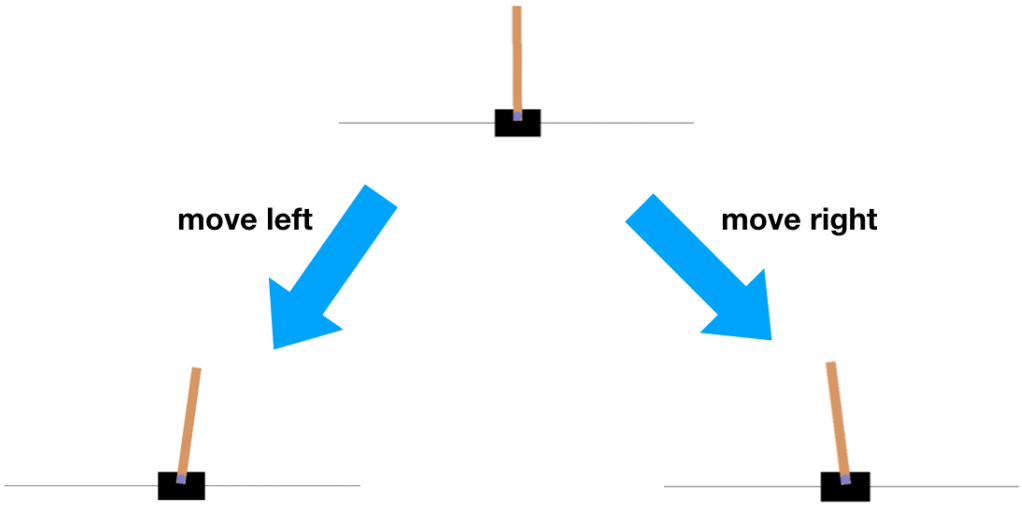


Figure 5.1: A graphical example of the dynamics of the CartPole environment. By moving the cart to either the left or right, the controller can prevent the pole from falling over.

Experimental Setup

The OpenAI Gym implementation of this environment was used here [110].

The NEAT algorithm (cfr. Section 2.5.2) was used to evolve the CPPN genomes, which the AESHN algorithm uses to build the candidate solutions (i.e. neural network controllers). These candidate solutions were evaluated using a fitness score. This fitness score was set to the number of time steps that the controller was able to keep the pole from falling over, i.e. the performance of the candidate solution as described above.

Population size was set to 300 with an *elitism*[1] of 2. CPPN mutation probabilities were set to 0.03 and 0.005 for connection additions and removals, 0.02 and 0.005 for neuron addition and removals and 0.94 for connection weight updates. By keeping the structural mutation probabilities rather low compared to the connection weight updates, we allow the evolved CPPN genomes some time to adapt before structurally mutating them.

As shown in Figure 5.2, the underlying CPPN genomes only use controller specific input and output nodes (given that there is no morphological evolution here). The substrate, i.e. the space in which we let AESHN evolve a neural network controller, is chosen to be a two-dimensional rectangle. This rectangle is centered at origin and covers the range $[-1, 1]$ in both the horizontal x and vertical y dimensions. Given the four input observations, we manually place four input neurons within this substrate on the horizontal $y = -1$ line. The two output (action) neurons corresponding to going left and right are placed on the horizontal $y = 1$ line. This setup is further illustrated in the paragraph below below.

1. The number of most-fit individuals in each species that are preserved as-is from one generation to the next

The variance thresholds used by the AESHN algorithm were set to 0.5 for the division phase, 0.2 for the pruning phase and 0.5 for the band-pruning phase. The minimum substrate resolution was set to 2, while the maximum was set to 3. This corresponds to the minimum and maximum depth of the quadtree representation of the hypercube (cfr. Section 2.5.4), leading to a maximal neural complexity of 64 neurons. The activation function within the evolved neural network controllers is set to the hyperbolic tangent function.

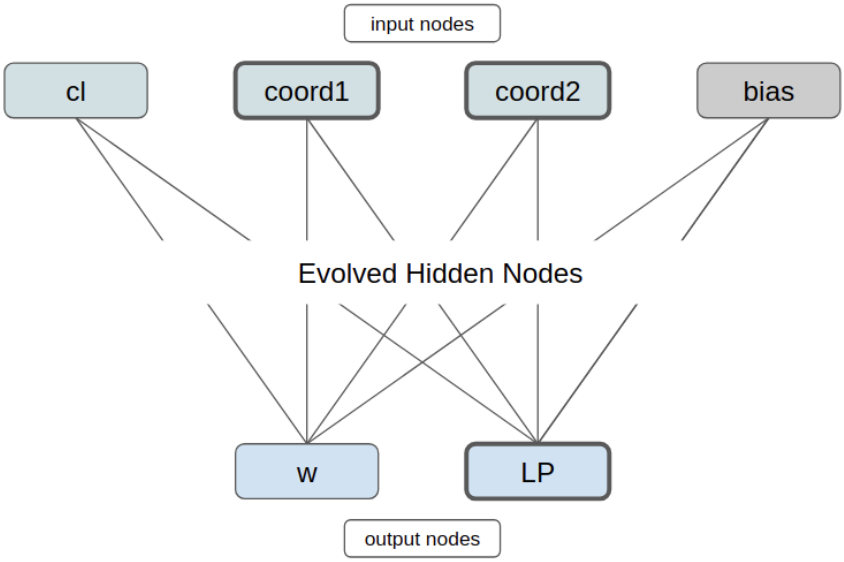


Figure 5.2: Illustration of the underlying CPPN architecture. The *coord1* and *coord2* input nodes are fed the coordinates of neurons within the substrate for which the output nodes denote the connection weight and learning parameters. The *cl* input node is given the euclidean distance between *coord1* and *coord2*, while the *bias* input node always receives +1.

Results and Discussion

The AESHN algorithm was able to solve this task rather quickly[2] (only 5 generations were necessary) as can be seen in Figure 5.3. Given the population size of 300, this is equivalent to total of 1500 evaluations[3].

2. Without multiprocessing, it took 7 minutes and 13 seconds on a Intel(R) Core(TM) i7-4790 CPU (3.60Ghz).

3. The number of evaluations is calculated as $generations \times population\_size$. While 1500 evaluations may seem much for this simple task, one of the main strengths of an evolutionary approach is its ability

The highest achievable fitness in this environment corresponds to the maximal number of time steps in one episode, which in this case is 500. One of the evolved neural network controllers that obtained this maximal fitness is shown in Figure 5.4. Given the user-defined input and output neuron locations within the substrate, the AESHN algorithm builds these neural network controllers by extracting hidden neurons and connections from the underlying CPPN produced patterns. This controller uses both standard and *neuromodulatory* connections. As described in Section 2.5.4, these neuromodulatory connections allow the network to change its connection weights during its lifetime, allowing it to adapt based on the input that its given.

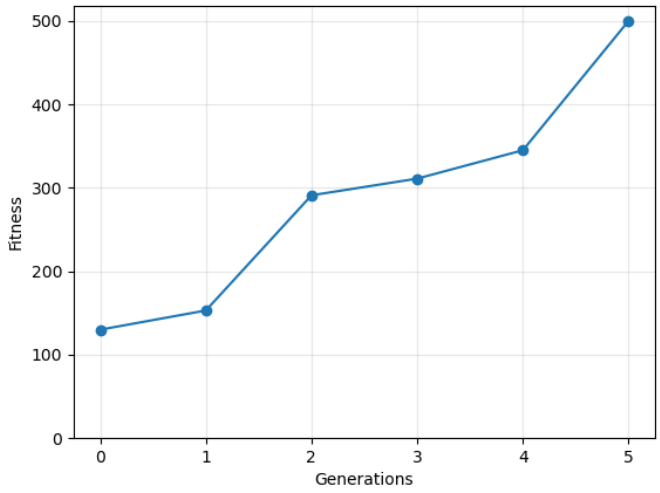


Figure 5.3: The fitness (y) of the best performing individual over every generation (x). Only 5 generations were needed to evolve a controller capable of achieving the maximal fitness of 500.

to be parallelizable. this means that depending on the number of cores available, we can often do multiple evaluations simultaneously.

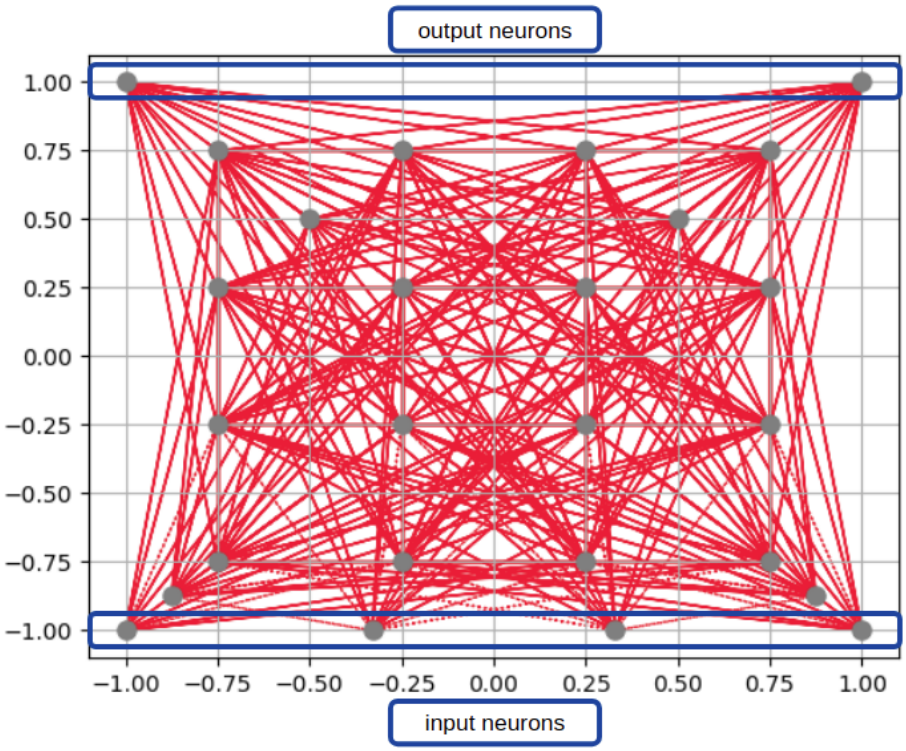


Figure 5.4: Visualisation of an evolved neural network controller, capable of solving the Cart-pole task. This neural network was evolved by the AESHN algorithm in a two-dimensional rectangular substrate. Connections are colored black if positive and red if negative, i.e. all connections within this network are negative. Standard connections are full lines, neuromodulatory connections are dotted lines. The four input neurons correspond (respectively from left to right) to the position of the cart, velocity of the cart, angle of the pole and angular velocity of the pole. The two output neurons correspond to the two actions, i.e. going left and going right. While at first sight it may seem strange that all connections are negative, this indicates that the controller works by always countersteering based on the movement observations of the cart and the pole that its given.

5.1.2 MountainCar

The Environment

Introduced in [111], the *MountainCar* environment contains a car placed on a one-dimensional track. The car is positioned between two "mountains" and the goal is to drive the car up the mountain on the right. This is illustrated in fig. 5.5. The difficulty of this environment lies in the fact that the car's engine is not strong enough to scale the mountain in a single pass. To succeed, it first has to drive back and forth to build up enough momentum. Performance of a solution here is measured as the number of time steps required in order to reach the target goal, i.e. the lesser time steps required the better.

Every time step, the controller is given only two input observations: (1) the position of the car along the one-dimensional track and (2) the velocity of the car. Given these observations, the controller has to choose between the three following actions: (1) accelerate to the left, (2) no acceleration, (3) accelerate to the right. This movement control is again done in a discrete manner, i.e. applying a force of -1 (left), 0 (neutral) or 1 (right).

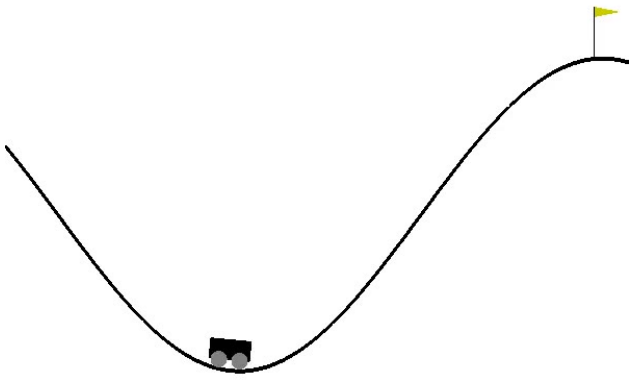


Figure 5.5: Graphical view of the MountainCar environment. The goal is to drive the car to the flag on the top of the right mountain. However, the car's engine is not strong enough to do this in one run. Therefore, the only way to succeed is to drive back and forth between the two mountains to build up enough momentum.

Experimental Setup

The OpenAI Gym implementation of this environment was used here [112].

The same NEAT and CPPN setup was used as in the CartPole experiment described above. The substrate is again a two-dimensional rectangle centered around origin, with in this case two input neurons corresponding to the observations placed on the $y = -1$ line and three output neurons corresponding to the actions placed on the $y = 1$ line.

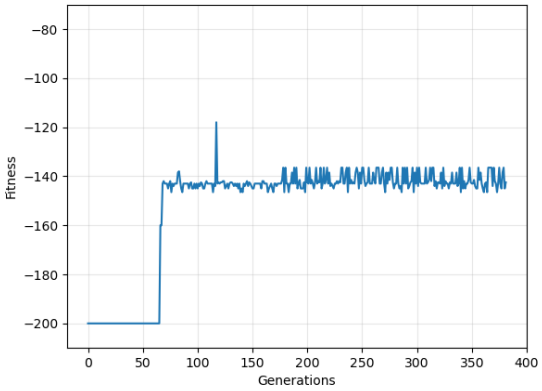
Two runs were done with this environment, one using a fitness score and one using the novelty approach discussed in Section 2.1.2. The fitness of a candidate solution is calculated according to the common standard for this environment, i.e. the negation of the number of time steps needed before reaching the target flag (with 200 as the maximum number of time steps). To calculate the novelty score of a candidate solution, we have to design a *behavior descriptor* that describes the candidate solution's functional behavior. In this case, the functional behavior can be seen as the trajectory that the controller makes the car follow and consequently the *behavior descriptor* here was set to 3 position samples taken from this trajectory (at equidistant points in time). The more the behavior descriptor of a candidate solution deviates from previously seen behavior descriptors, the higher the novelty score it is given.

Given the sparsity of the fitness score used here, the hypothesis is that the novelty approach will better guide the evolutionary search towards solving this environment. While it does not directly optimize towards solving the task, it does stimulate the generation of controllers that drive the car towards all possible locations. Given that the target location is one such possible locations, it should be able to evolve a controller that reaches this target.

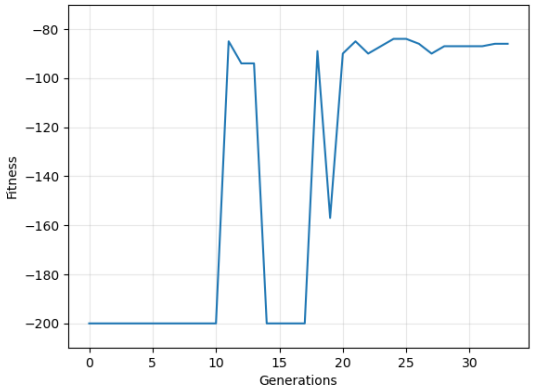
Results and Discussion

In line with the hypothesis given above, the novelty search approach indeed led to better results. One the one hand, it more quickly evolved controllers that solve the environment (requiring only 11 generations instead of 67 generations for the fitness based approach, equivalent to respectively 3300 and 20100 evaluations given the population size of 300 here). One the other hand it evolved controllers that better solve the environment (requiring only 84 time steps to reach the goal), while the fitness based approach stagnated. This is shown in Figure 5.6. The main reason for this is that, as described in Section 2.1.2, the novelty approach allows to reward important stepping stones such as first driving the car to the left (which in turn allows to build up momentum), while the proposed fitness based approach does not. While other fitness score calculations were also possible, such as calculating it based on the remaining distance between the car and the goal, none of them allow to encapsulate the dynamics of first building up momentum. As the novelty search approach steers away from this "goal driven" paradigm, the additional benefit is that the user no longer has to worry about trying to encapsulate these environment dynamics within a score function.

Next to the performance of the evolved neural network controllers, its also interesting to take a look at their structure within the substrate. In Figure 5.7, two such controllers are shown which were both evolved using the novelty search approach. This figure showcases how the neuroevolution algorithm was able to reduce neural complexity by evolving a different kind of neuron next to the standard neurons: recurrent neurons. Next to connections to other neurons, a recurrent neuron has a connection to itself. This allows the neuron (and by extent the ANN) to have a simple form of memory. As having memory is quite advantageous within this MountainCar environment, the recurrent neurons allowed to reduce the neural complexity while maintaining the same level of performance.



(a) Fitness based approach



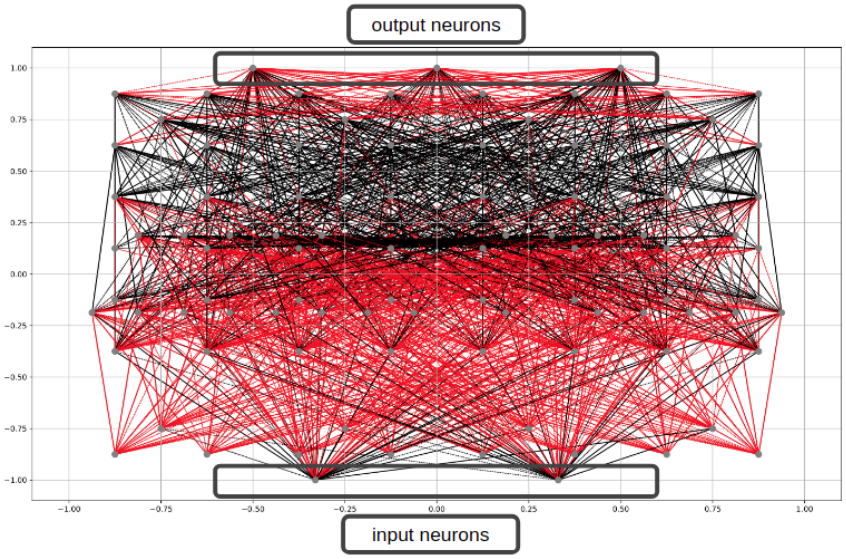
(b) Novelty based approach

Figure 5.6: These two figures depict the fitness of the best individual of each generation. The fitness is defined as the negation of the number of time steps required to drive the car to the target location. While the fitness based approach (a) first achieves this at generation 67 (evaluation 20100), the novelty approach already succeeds at generation 11 (evaluation 3300). Additionally, the novelty approach was able to evolve better controllers that more quickly drive the car towards the target location.

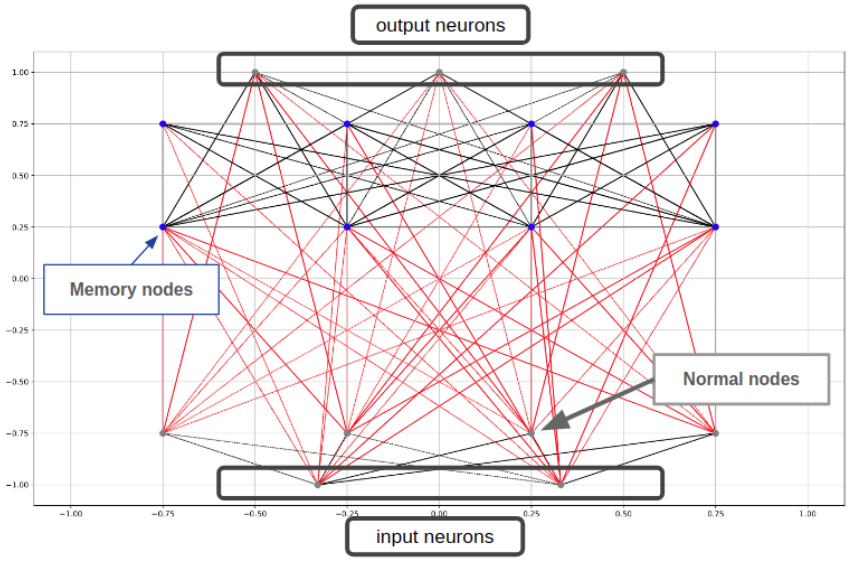
5.2 Towards Moving Creatures

Given the positive validation of the neuroevolution algorithm in Section 5.1, this series of experiments incorporates the morphological evolution and thereby applies the complete complete *One CPPN to Rule them All* (OCRA) approach proposed in Chapter 3. Within the proposed OCRA approach, the main idea is to represent both agent morphology and controller by the same underlying genome, the Compositional Pattern Produced Network (CPPN). The key insight is that when evolving only a single genome that represents both controller and morphology, this should more naturally incorporate the simultaneous evolution of the two and thereby mitigate the current issues of simultaneous evolution of morphology and control. The OCRA approach thereby establishes our main strategy to achieve research objective one, described in Section 1.1.

As described in Chapter 1, *virtual creatures* are chosen to be the entity on which the pro-



(a) Best performing controller at generation 11.



(b) Best performing controller at generation 24.

Figure 5.7: Two neural network controllers capable of driving the car to the target location, evolved using the novelty search approach. (a) shows the first ANN controller that solved the environment. (b) shows a second ANN controller that was also capable of solving the environment but was able to do so with a much lower neural complexity. The main reason for this is that it evolved a different kind of neuron, next to the standard (grey) neurons, namely recurrent neurons (blue). Recurrent neurons allow the ANN to have some kind of memory by connecting to themselves. While much less complex, the ANN shown in (b) achieves the same level of performance as the ANN shown in (a).

posed techniques will be applied. Virtual Creatures are a type of embodied agent studied originally in the artificial life domain. Given that they require both a morphology and an artificial brain that enables to control that morphology, they establish an interesting object to study simultaneous evolution of brain and morphology with. By extent, this also has a strong link with the robotics domain.

The most basic requirement of such virtual creatures is that they are capable of moving around and that is exactly the goal of this series of experiments: evolving walking virtual creatures.

The Environment

Given the simplicity of this experiment, in essence we require nothing more than a simple plane on which the evolved (three-dimensional) virtual creatures can walk around. Nevertheless, we require a simulator with a good backbone physics engine. The simulator used throughout this thesis is Unity [113]. Given the release of the Unity ML-Agents Toolkit [114], it provides a good interface for using external optimization techniques to train simulated agents. Thereby it allows for the creation of modular robots such as the virtual creatures proposed here. Additionally, Juliani et al. [115] argue that modern game engines are uniquely suited to act as general platforms and as a case study examine the Unity engine and Unity ML-Agents Toolkit, demonstrating its advantages.

The environment is shown in Figure 5.8. Therein, each evolved virtual creature will be placed in the middle of the yellow plane, their only goal being to move as far away from that starting location as possible in a limited number of time steps. Consequently, performance is measured as the maximum distance that the virtual creature was able to move from its original position.

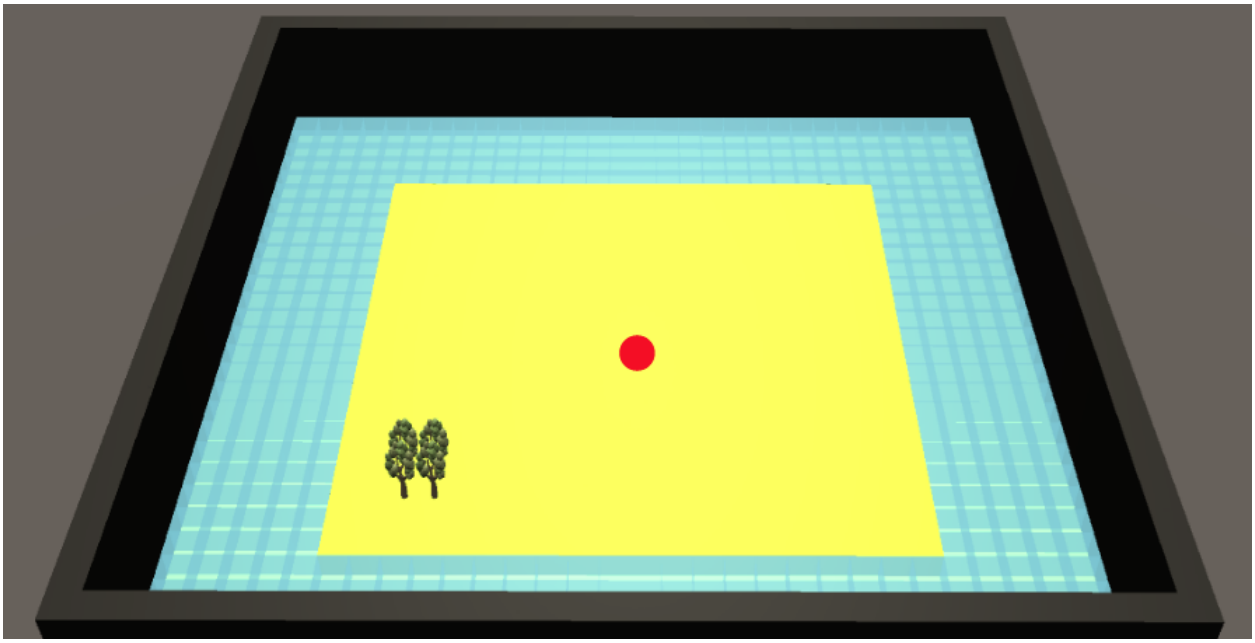


Figure 5.8: The environment in which the moving creatures are evolved, made in Unity [113]. Each evolved virtual creature is initially placed at the middle of the yellow plane (red dot). Their only goal is to move as far away from that starting location in a limited number of time steps. This figure also poses as a small preview of the Virtual Ecosystem discussed in Part II of this thesis, wherein the yellow plane represents the earth containing (green) vegetation and which is surrounded by water.

Experimental Setup

The NEAT algorithm (cfr. Section 2.5.2) was used to evolve the CPPN genomes, which are here used to represent complete virtual creatures. Population size was set to 120 with an elitism of 2. CPPN mutation probabilities were set to 0.03 and 0.005 for connection additions and removals, 0.02 and 0.005 for neuron addition and removals and 0.94 for connection weight updates. By keeping the structural mutation probabilities low, we allow the evolved CPPN genomes more time to adapt before they are structurally mutated. Given that the proposed OCRA technique is applied here, the CPPN architecture discussed in Section 3.3 is used.

Within the OCRA approach, the *Brain Substrate* (the space in which we let AESHN evolve a creature's neural network controller) is chosen as the *brain block* within the modular creature morphology (cfr. Section 3.2). Consequently, the evolved neural network controllers are now three-dimensional. The variance thresholds (VT) used by the AESHN algorithm were set to 0.5 for the division phase, 0.2 for the pruning phase and 0.5 for the band-pruning phase. The minimum substrate resolution was set to 2, while the maximum was set to 3. This corresponds to the minimum and maximum depth of the quadtree representation of the hypercube (cfr. Section 2.5.4), leading to a maximum neural complexity of 512 neurons. The activation function within the evolved neural network controllers was set to the hyperbolic tangent function.

Next to the morphology dependent input and output neurons, the neural network controllers are fed three additional inputs through *general input neurons*: a constant bias (+1), along with a time dependent sine and cosine wave. Given that the AESHN algorithm uses the underlying CPPN genome's patterns to choose which input neurons to connect, for each such additional input it can thus evolve towards either using them if they prove to be useful

or evolve towards ignoring them. This means that no true harm can be done by just making them available (besides a minor increase of complexity coupled with with a potential increase of training time). Providing such oscillatory input signals to neural network controllers has proven to be useful within the robotic gait learning domain [19]. The main reason for this is that they stimulate periodic gaits. Therefore, we expect them to be used by the evolved controllers.

On the morphological side, only the *filler blocks* and their potential *joint connections* (leading to *joint blocks*) were made available. Given that there is no real need for sensors here, the *sensor block* was not made available to reduce the size of the search space and potential complexity of the evolved virtual creatures. The *morphological resolution* was set to 5, i.e. the three-dimensional *Agent Space* cube is subdivided into 5 subcubes in each dimension or into 125 subcubes in total.

The novelty search approach was used as the evolutionary search algorithm (cfr. Section 2.1.2). Given that complete virtual creatures are evolved here, the *behavior descriptor* has to incorporate both the morphology and the control.

To describe the morphological behavior, we subdivided the three-dimensional Agent Space into 8 quadrants. Each of these quadrants were then described by 2 numbers that denote the (normalized) amount of filler blocks and joint blocks present. This results in a morphology descriptor of size 16.

The functional behavior of the controller was quantified using 5 creature position samples taken at equidistant points in time. This represents the trajectory of the virtual creature and thereby constitutes a simplified representation of the functionality of the controller. These position samples are the two-dimensional coordinates of the creature on the ground plane (the vertical position is not taken into account) and were again normalized to lie in the range

of 0 to 1. This results in an additional 10 numbers for the controller side of the behavior descriptor, totalling up to a behavior descriptor of size 26.

As a final note, the experiment was run on four *Intel(R) Xeon(R) CPU E5-2650 v2 octa-core @ 2.60GHz* CPUs, using multiprocessing to take advantage of the 32 cores available. Table 5.1 summarizes the experimental setup used for the results discussed below.

CPPN mutation probabilities	Neuron addition / removal	0.02 / 0.005
	Connection addition / removal	0.03 / 0.005
	Connection weight update	0.94
AESHN	Division phase VT	0.5
	Pruning phase VT	0.2
	Band-pruning phase VT	0.5
	Min / Max substrate resolution	2 / 3
Evolutionary Search	Algorithm	Novelty Search
	Population size	120
	Behavior Descriptor	Trajectory + morphology
Computational Setup	CPU	4 x Intel XeonCPU E5-2650
	Total number of cores	32

Table 5.1: Experimental setup of the "Towards Moving Creatures" experiment.

Results and Discussion

Somewhat to our surprise, the first run already resulted into virtual creatures capable of walking a distance of up to 20 times their own morphological size. Figure 5.9 shows the

highest relative[4] distance a creature was able to travel for each generation. This figure also shows that the first virtual creature capable of moving was already evolved after 7 generations (equivalent to 840 evaluations given the population size of 120). This corresponds to 19 minutes and 11 seconds wall clock time using multiprocessing on the 32 core setup mentioned above. The best performing virtual creature[5] was evolved at generation 69 or evaluation 8280 (corresponding to 11 hours and 49 minutes wall clock time using multiprocessing on the 32 core setup mentioned above.). This creature was able to move a distance equal to 20.07 times its own size in the given time frame (2000 steps) and its morphology is shown in Figure 5.10a. Next to this creature, others[6] such as the snake-like creature shown in Figure 5.10b were evolved as well, although moved somewhat slower.

To inspect the ability of the mutated CPPNs to represent both valid morphologies and ANN based controllers, we can take a look at the percentage of valid creature morphologies and controllers within each generation's population. This is shown in Figure 5.11. In this environment, morphologies are invalid if they have no *joint block*, given that this means that they are not capable of movement. In line with the validity of a morphology, a neural network based controller is invalid if it does not connect to any of the output neurons corresponding to a *joint block*. Given that the morphology is always created first (this allows to locate the morphology dependent input and output neurons), the controller will not be created if the morphology is invalid. This allows to save some computation and is the reason why the ratio of valid controllers, shown in Figure 5.11b, is not the percentage of valid controllers of the whole population, but only of those which have valid morphologies.

4. Given that the size of a creature's morphology may impact the distance it is able to travel in the limited number of time steps, we divide its distance traveled by its maximum size in any of the x, y and z dimensions.

5. A video demonstrating its movement is available at: https://www.shorturl.at/dmoCJ

6. A video demonstrating the diversity and progress of creatures during the first 10 generations of the evolutionary search is available at: https://www.shorturl.at/wCGNO

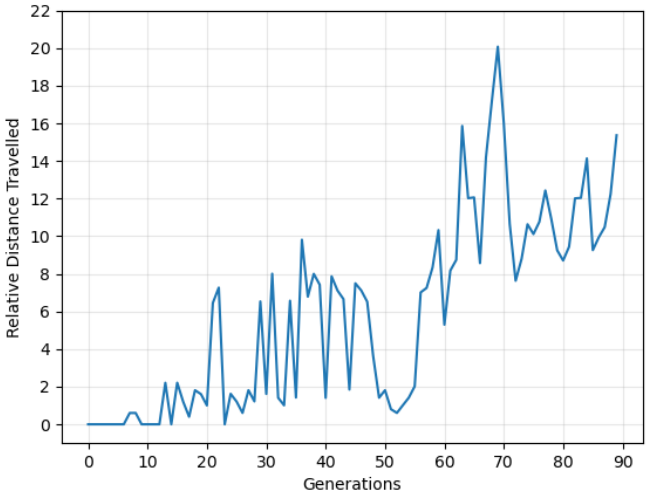
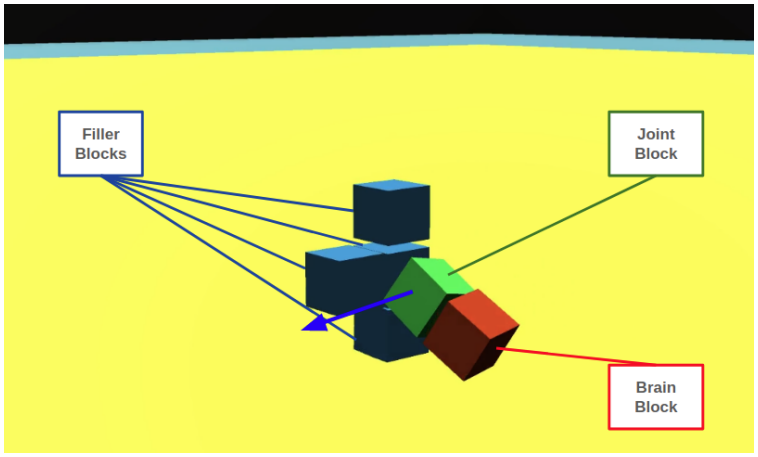


Figure 5.9: The furthest distance travelled relative to creature size from the initial starting location (y) over every generation (x). After 7 generations, the first virtual creature capable of moving was evolved. The best performing virtual creature was evolved at generation 69 and moved a distance equal to 20.07 times its own size.

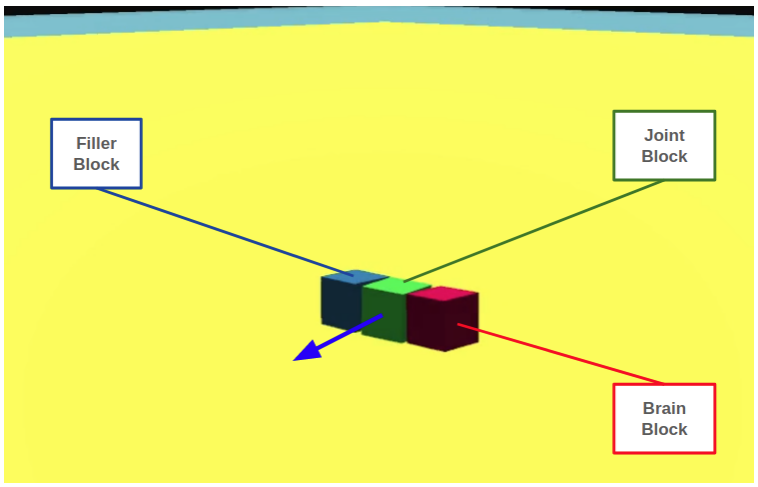
The figure shows that, as evolution progresses, the underlying CPPNs are mutated in such a way that they better and better are able to represent both valid morphologies and controllers.

We can also take a look at the percentage of (valid) creatures that make use of the provided oscillatory sine and cosine signals. As shown in Figure 5.12, these signals are nearly always incorporated by the evolved neural network controllers. This indicates that these additional inputs do provide some advantage, as the CPPN genomes that lead to connecting them would otherwise be discarded during the evolutionary search. These results are in line with the work of [19], wherein the positive influence of such oscillatory input signals when using the AESHN algorithm to evolve controllers for gait learning has already been investigated.

Figure 5.13 visualizes the average complexities of the evolved controllers (phenotypes) in function of the average performance that they lead to. As described above, performance in this case is measured as the relative distance a creature was able to move away from its

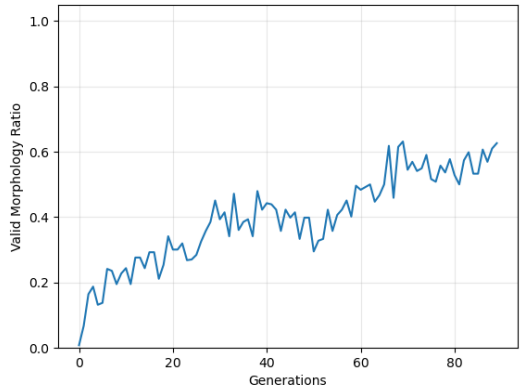


(a) The best performing virtual creature evolved at generation 69. This creature moved a distance equal to 20.07 times its own size. This creature is able to walk by using the joint block to periodically lift its brain block, move it forward and put it down as an anchor to drag the rest of its body (the filler blocks) forward.

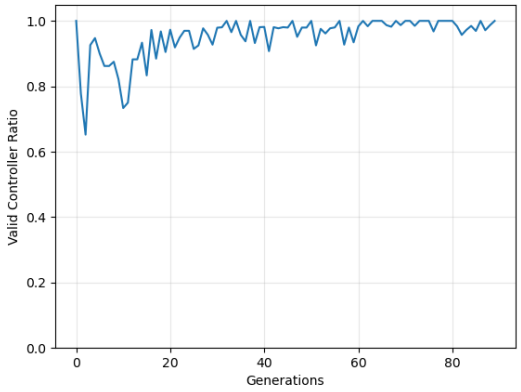


(b) An alternative virtual creature evolved at generation 36. This creature moved a distance equal to 9.81 times its own size. This creature is able to move by using the joint block to periodically switch between lifting and moving its filler block and its brain block.

Figure 5.10: Two examples of moving creatures evolved by the OCRA approach. The different building blocks of the creature morphologies are annotated and a blue arrow denotes the direction in which they were observed to move in.

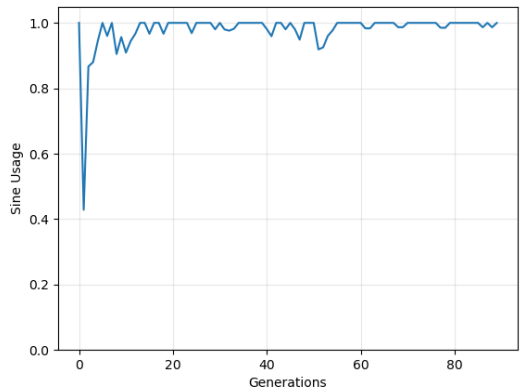


(a) Ratio of valid morphologies over population size (y) within each generation (x).

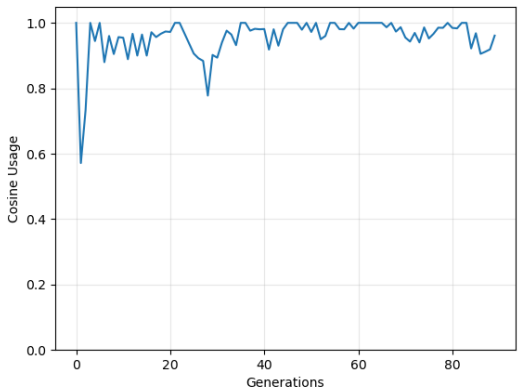


(b) Ratio of valid controllers over the amount of valid morphologies (y) within each generation (x).

Figure 5.11: The percentages of valid creature morphologies (a) and controllers (b). A morphology is invalid if it does not contain a *joint block*. A controller is invalid if all output neurons corresponding to *joint blocks* are unconnected. This figure shows that, as evolution progresses, the mutated CPPN genomes become better and better at representing both valid morphologies and controllers.



(a) Percentage of valid creatures that use the sine signal (y) per generation (x).



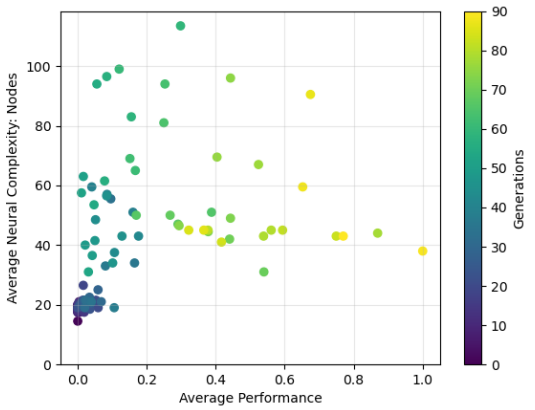
(b) Percentage of valid creatures that use the cosine signal (y) per generation (x).

Figure 5.12: The percentage of valid creatures of which the evolved controllers made use of the oscillatory sine (a) and cosine (b) signals. These were made available to the controller through *general input neurons*. As can be seen, these are more often than not incorporated within the controllers. This indicates that they do provide some advantage.

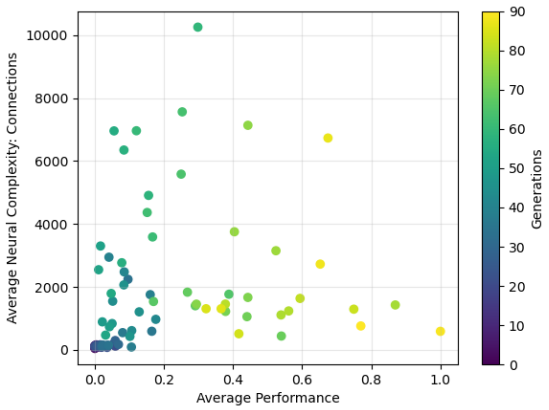
starting location. Figure 5.13a shows this in terms of the average amount of hidden nodes in the ANN based controllers, while Figure 5.13b denotes this in terms of the average amount of connections. Figure 5.13c then merges these two types of neural complexities by first normalizing them (i.e. scaling them to the range of 0 to 1) and taking their average. These figures also incorporate the generation at which the average neural complexity and performance were measured through a color gradient.

The two major insights gained from this are: (1) the neuroevolution covered a large range of controller complexities in its search towards a fitting complexity for the problem at hand and (2) while initially diversifying, over time the neural complexities converge (towards an average of around 40 hidden nodes and 1000 connections) leading to an increase in average performance.

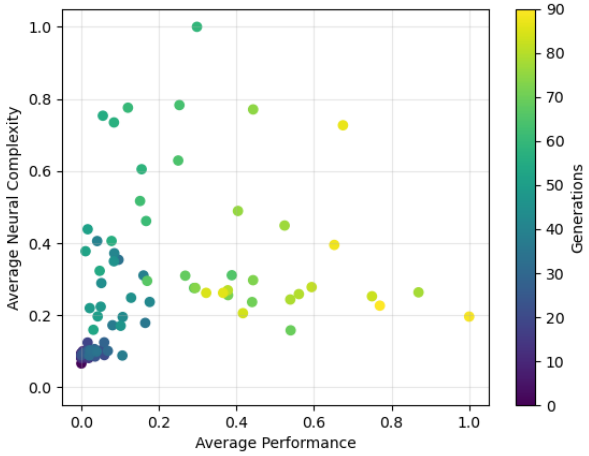
Next to the average neural complexity of the resulting phenotypic controllers, we can also take a look at the average neural complexity of the underlying CPPN genomes. Given that these are of the indirect encoding type, it is hard to directly correlate the genotypic complexity with the resulting phenotype's performance in a useful manner. However, we can analyse the relation between the complexity of the genome and the complexity of the phenotype that it represents. This allows us to gain insights about the encoding strength of the genome, i.e. the level of complexity it is able to represent given its own level of complexity. While in the OCRA approach the phenotype corresponds to a complete virtual creature and thus to both a morphology and neural network controller, we will focus on the relation with the complexity of the controller. Similar to the discussion of the controller complexity in function of performance given above, Figure 5.14 visualizes the phenotypic controller complexity in function of the genotypic CPPN complexity. Next to giving an overview of the absolute amount of hidden nodes and connections within both the CPPN and resulting con-



(a) Average amount of hidden nodes in function of average performance.



(b) Average amount of connections in function of average performance.



(c) Average neural complexity (mean of normalized node and connection complexity) in function of average performance.

Figure 5.13: Visualizations of average neural complexities of the evolved controllers (phenotypes) in function of their average performance, with color denoting the generation. Performance here is the relative distance that the creature was able to travel from its starting location. (a) shows the neural complexity in terms of the amount of hidden nodes, while (b) shows it in terms of amount of connections. (c) then merges these two types of neural complexities into one by taking the average of both the normalized amount of nodes and connections. These figures show at what controller complexity the best performance was measured, while simultaneously providing insights about the evolved complexity over time.

troller in Figure 5.14a and Figure 5.14b respectively, Figure 5.14c again merges the node and connection complexities into a single neural complexity for both the genotype and phenotype.

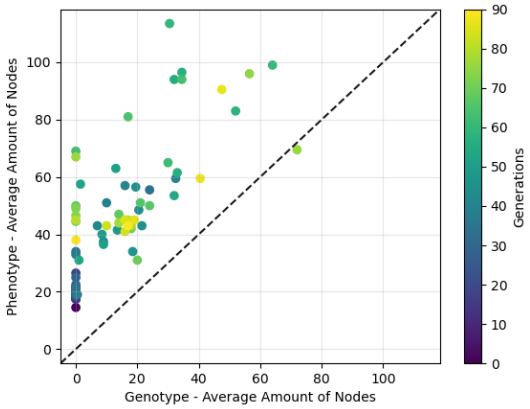
The main insight here is that the CPPN genome indeed remains simpler than its resulting controller as previously discussed in Section 2.5.3. This is indicated in the figures through a diagonal dotted line that represents equal genotypic and phenotypic complexity. Given that most of the complexity samples are above this line, the phenotypic complexity is therefore often higher compared to the genotypic complexity. This, combined with the fact that the same CPPNs also represent the entire morphology substantiate its encoding capabilities[7]. Additionally, the underlying quadtree, which represents the hypercube of connection weights within the Brain Substrate, was limited to a maximum depth of 3. This directly limits the maximal attainable phenotypic controller complexity as this sets the maximal number of target neuron locations that a given neuron can be connected with (cfr. Section 2.5.4). However, as demonstrated by the authors in the original ES-HyperNEAT paper [92] (but not tested here), the same CPPN genomes evolved at this lower Brain Substrate resolution can represent similarly performing controllers at a higher resolution and consequently at a higher phenotypic complexity as well. The main reason being that CPPN genomes are capable of producing a theoretically infinite-resolution pattern of weights within the hypercube, while we only sample this pattern at a limited resolution and therefore only allow a limited controller complexity.

The somewhat positive correlation between increasing genotype and phenotype complexities is also to be expected. As increasing genotypic complexity allows for increasingly complex output patterns, these patterns are likely to contain more variance. Given that the AESHN algorithm is used to create neural network controllers, a higher variance causes more neurons

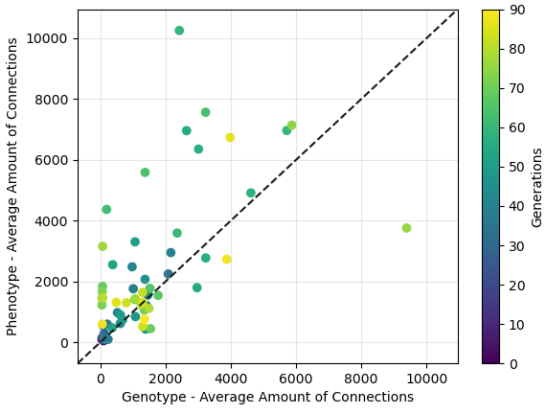
7. The CPPNs encoding capabilities are further substantiated by the results at the end of the next series of experiments, described in Section 5.3.3.

and connections to be placed within the *Brain Substrate* (cfr. Section 2.5.4), leading to a higher phenotypic complexity.

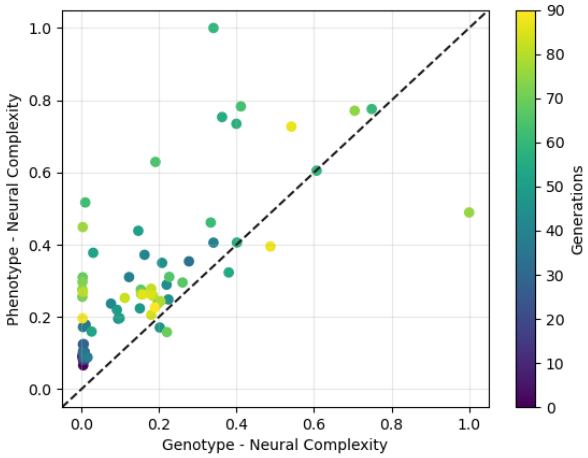
In conclusion, this experiment can be seen as an initial validation and proof of concept of the proposed *"One CPPN to Rule Them All (OCRA)"* approach. Next to demonstrating its ability to evolve moving creatures, some analyses were made of (1) the capability of the CPPN genome to represent both valid morphologies and controllers, (2) the neural complexities of the resulting controllers and (3) the encoding strength of the CPPN genome. While taking a look at the structure of the evolved CPPN genomes and at the spatially organized ANN controllers embedded within the morphology is also interesting, this is will not be done here but instead will be investigated in the next series of experiments.



(a) Average controller complexity in function of the average CPPN complexity, in terms of the amount of hidden nodes.



(b) Average controller complexity in function of the average CPPN complexity, in terms of the amount of connections.



(c) Average neural complexity (mean of normalized node and connection complexity) of the controller in function of the neural complexity of the CPPN.

Figure 5.14: Visualizations of the average neural complexities of the phenotypic controllers in function of the average neural complexities of the underlying CPPN genomes. Color denotes the generation in which the average complexity samples were taken. (a) shows the neural complexity in terms of the amount of hidden nodes, while (b) shows it in terms of the amount of connections. (c) then merges these two types of neural complexities into one by taking the average of both the normalized amount of nodes and connections. These figures mainly indicate the representative power of the CPPN genome, allowing it to encode much more complex phenotypic results in comparison to its own complexity. This is further emphasized through the diagonal dotted lines that represent equal genotypic and phenotypic complexity.

5.3 Towards Seeing Creatures

Given the positive validation of the OCRA methodology to completely evolve moving creatures, this series of experiments incorporates the usage of sensors within these creatures. Therefore, the creatures are evolved in environments that contain visible target objects to which they must move. To do so, the creatures must use their sensors to identify and locate these targets.

In Section 5.3.1 we first evaluate the ability of the controllers to process *sensory-like* observations by manually feeding the direction and distance to the target location to predefined input neurons. In other words, we provide the creatures with *extrinsic* guidance towards their targets. Next, the experiments in Section 5.3.2 allow creatures to evolve sensor blocks themselves, consequently making them responsible for evolving their own set of observations and making the shift from *extrinsic* to *intrinsic* guidance. While neither of those experiments were completely successful, their failure leads to some useful insights concerning the OCRA methodology in general and more appropriate environment designs. In light of these insights, the experiments discussed in Section 5.3.3 reduce complexity by simplifying both the creatures and the environment, thereby evolving creatures that successfully exhibit functional sensor usage.

5.3.1 Extrinsic Guidance Experiment

Given that immediately incorporating *sensor blocks* within the evolutionary process would introduce quite a big increase in complexity, this experiment first validates an important intermediate requirement of the controller. This intermediate requirement is its ability to processing *sensory-like* observations in such a way that it is able to react to these observations by appropriately steering the morphology. Particularly in this series of experiments, these observations correspond to certain target locations to which the creature must move. If this

was to be done by letting the evolutionary process place *sensor blocks* within the creature morphologies, this would require the evolutionary process to (1) create movable creature morphologies containing sensor blocks with functional[8] configurations and (2) simultaneously evolve controllers robust enough to deal with these additional sensory observations. Furthermore, the input neurons corresponding to the sensor blocks can be anywhere within the *Agent Space*, as the sensor blocks can be placed anywhere.

This experiment sidesteps this sensor block related complexity by providing the necessary observations to the controller manually, i.e. we give the creature *extrinsic guidance* towards its goal. This is done through four user-defined *general input* neurons (cfr. Section 3.2), which are fed the three-dimensional (x, y, z) direction towards the target location and the distance to that target location.

The Environment

The environment is mainly the same as the one used to evolve walking creatures in Section 5.2 and was again created in Unity. The only difference here is that there now exists some randomly generated target location to which the creature must move. The target locations are random within every evaluation and are always generated at some distance from the creature's current location. This distance is made dependent on creature size. If a creature reaches a target location, a new one is spawned in such a way that there is a minimum and maximum difference of 45 and 90 degrees respectively between the previous direction the creature had to move in and the direction towards this new target location. This is illustrated in Figure 5.15. The reason for this is that it avoids creatures that reach multiple target locations by just walking in one direction, thereby stimulating the evolution of creatures that are more flexible in their movement. Performance in this environment is measured by

8. To be functional, a sensor block must be placed at the exterior of the morphology and must point outwards.

the amount of target locations a creature can reach within a limited amount of time steps (4000 in this case).

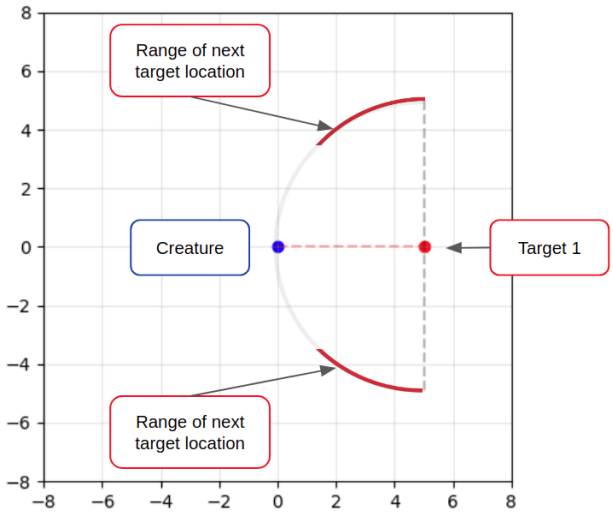


Figure 5.15: A two-dimensional example of where the second target location may spawn after the creature (blue) has reached the first target location (red). New target locations are placed randomly, with the constraint that there must exist a minimum and maximum difference of 45 and 90 degrees respectively between the previous direction and the new direction towards the new target.

Failed Runs

In the first initial test runs, this experiment used exactly the same experimental setup as the previous experiment discussed in Section 5.2. Consequently, novelty search with the same behavior descriptor (incorporating the creature's morphological configuration and trajectory) was applied. None of these runs however led to any significant results within this environment. While they were all again capable of evolving moving agents, none of them was able to evolve a creature that reached more than one target location. Furthermore, the first target location was observed to mainly be reached by coincidence as some creatures had the luck that their initial target location spawned in the same direction as they always travel (the mechanism shown in Figure 5.15 was not yet incorporated here).

Afterwards, the environment was somewhat adapted such that an indirect link between a higher performance and a higher novelty score was incorporated. Given that in the previous setup there was no explicit reason why optimizing (or evolving) towards novel creatures would eventually cause creatures to start moving in the directions we give them[9]. This link was incorporated by early stopping a creature's evaluation if the creature started to move away from its target goal instead of approaching it. Because of that early stopping, a creature's trajectory would not be complete and part of its behavior descriptor would be filled with some default value. This in turn would cause a lower novelty score, given that many creatures' behavior descriptors would contain that default value. This means that in order to maximize novelty, creatures had to be evolved that moved towards the target locations. Again, this setup did not lead to any significant results.

Some experiments using a fitness based approach were also conducted. Fitness is defined here as the amount of target locations a creature was able to reach during its (bounded) lifetime, added to the inverse of the remaining distance to the latest goal (that the creature was not able to get to). These fitness based experiments led to similarly negative results as the initial novelty search approach. While the negative results of the fitness based experiments could be explained by the noisiness of the fitness function (target locations are spawned randomly, which is in favor for some agents but disadvantages others), it does not explain why the novelty search approach failed as well.

In hindsight, the reason why the novelty approach with early stopping also failed is probably because instead of introducing a noisy fitness function, we introduced a noisy novelty

9. While this may sound strange, given that the fundamental idea behind the novelty search approach (cfr. Section 2.1.2) is to *"abandon the objective function"*, there must be some minimal link between novelty and performance as otherwise the chance of evolving an actual solution to the problem is quite small.

function by making it dependent on the randomly generated target locations. Consequently, we stimulated the creatures towards evolving a "sweeping strategy", i.e. how the creatures need to move to most likely pick up the targets blindly. There must however still exist some link between novelty and performance. In essence, the better approach here would have been to find a better way to incorporate "the act of moving towards target locations" within the behavior descriptor, i.e. devising a more appropriate behavioral encoding that is more expressive in regard to the target. This could for example be done by making the trajectory samples within the behavior descriptor dependent on the target location (e.g. treating the target location as the origin point when tracking the creatures position through coordinates). Additionally, the noisiness in the fitness based approach could have been mitigated by evaluating creatures in two separate trials and then aggregating the fitness results of those two separate runs, for instance in a strict manner by taking the minimum of the two. This would avoid insignificant creatures being rewarded with high fitness score.

Nevertheless, at the time we hypothesized that these negative results purely originated from a discrepancy between our environment and our evolutionary search algorithm. Given that the novelty and fitness based approaches both rely on creating an entirely new[10] population in every generation based on the population of the previous generation, we thought that the randomness of the environment resulted in important creatures being discarded from the population. Consequently, this would restrain the evolutionary process from leading to significant solutions.

Towards (Flawed) Success

This led to a new experiment in which we introduced the MAP-Elites algorithm (cfr. Section 2.1.3) to our architecture. The idea was that it would mitigate the issue described

10. Except for a user-defined number of elites that is transferred without change.

above, given that this algorithm steers away from a "dynamic population". It does so by introducing an archive in which we store the current best performing solutions (or elites) and from which we select the parents that are used to generate the offspring for the next generation. After evaluating this (mutated) offspring, their recorded (phenotypic) behavior descriptors allow to map each of their underlying CPPN genome to a specific archive cell. Such a genome is then only placed within that archive cell if the cell is still empty or if its phenotype performed better than the phenotype of the genome currently stored. Or in other words, if it has a higher fitness score. Fitness here was again defined as the amount of target locations a creature was able to reach during its (bounded) lifetime, added to the inverse of the remaining distance to the latest goal (that the creature was not able to get to). Additionally, within this new experiments we also introduced the notion of *build rewards*. This allows to differentiate between the genomes that represent a valid genotype and those that do not. Therefore every genome initially starts with a fitness of -2, which is increased to -1 if it results into a valid morphology and is increased to 0 if the controller is valid as well.

Using this archive of elites mitigates the risk of completely discarding well performing creatures which were just "unlucky" within their evaluation, given that their parent would still be stored in the archive. Although the main reasoning behind the introduction of this algorithm was flawed (we should have focused on a better behavior descriptor), it did however lead to solving this environment and resulted in a creature capable of reaching two randomly generated target locations within multiple independent evaluations (discussed in the results section below).

Experimental Setup

Specifically, the *Centroidal Voronoi Tessellations* (CVT) extension of the MAP-Elites algorithm proposed in [46] (cfr. Section 2.1.3) was used in order to avoid having to specify

an appropriate discretization of the underlying behavior descriptor ourselves, with the additional benefit that it allows us to explicitly set the number of cells within the archive. Consequently, this allows us to explicitly handle the quality-diversity trade-off of the algorithm. This extension approximates the Centroidal Voronoi Tessellation of the *behavioral space* by running Lloyd's (K-Means) clustering algorithm [116] on a set of behavior descriptors. Each cluster thereby corresponds to an archive cell and consequently the user-defined number of clusters (typically referred to as k) allows to set the number of archive cells. Within this experiment, k was set to 1000 and the archive was initialized using 100000 randomly generated behavior descriptors.

We also used a slightly different behavior descriptor in which the trajectory is no longer described by position samples but by direction samples, i.e. samples that indicate towards which direction the creature was moving at that point in time[11]. This limits the dependence of the trajectory on the creature's morphology[12], given that larger creatures may be able to move further than smaller creatures in between taking samples while still moving in the same direction. Additionally, this independence of creature morphology makes it easier to randomly generate the trajectory component of the behavior descriptors used to initialize the CVT-MAP-Elites archive, as described above.

In this experiment, only the mutation and crossover operators of the NEAT algorithm (cfr. Section 2.5.2 were used, given that we now employ the MAP-Elites framework as the evolutionary algorithm. CPPN mutation probabilities were set to 0.94 for connection weight

11. Again in hindsight, this behavior descriptor is badly designed as well. Given that creatures should move based on environmental input (randomly spawned target locations), their behavior should vary with respect to the environment. Consequently, this behavior descriptor is again not expressive with relation to the target.

12. The morphology is already detailed in a different part of the behavior descriptor, so its better to describe the trajectory in a morphology independent manner to avoid the morphology defining the behavior too much.

updates and 0.25 for all structural mutations (connection additions and removal, node addition and removal). Compared to the previous experiment (in which we used 0.03, 0.005, 0.02 and 0.005 respectively for these structural mutations) this is an order of magnitude higher. This is possible because we can rely on the MAP-Elites framework to keep (the CPPN genomes of) our best solutions stored within the archive, meaning that they will not be discarded by bad structural mutations like they would have in the previous algorithm that worked with a "dynamic population". Therefore, it allows for a more aggressive mutation strategy.

The same AESHN variance parameters were used as in the experimental setup discussed in Section 5.2. Given the positive validation of the supplementary *general input neurons* that are fed oscillatory sine and cosine signals in the previous experiment, they are provided to the controllers in this experiment as well. As described above, four additional *general input neurons* were added as well. These provide the controller with the observations necessary to be able to move to the current target location (given that no sensors are made available here): the three-dimensional direction towards the goal and the current distance towards that goal.

This experiment (and the failed ones above) were all run on the same computational architecture as the experiment described in Section 5.2: four *Intel(R) Xeon(R) CPU E5-2650 v2 octa-core @ 2.60GHz* CPUs, using multiprocessing to take advantage of the 32 cores available. We chose to make the population size dependent on the number of cores available as a way to make sure that every core was maximally taken advantage of. This resulted into a population size of 128 or four times the number of cores available. Table 5.2 summarizes the experimental setup used for the results discussed below.

CPPN mutation probabilities	Neuron addition / removal *	0.25 / 0.25
	Connection addition / removal *	0.25 / 0.25
	Connection weight update	0.94
AESHN	Division phase VT	0.5
	Pruning phase VT	0.2
	Band-pruning phase VT	0.5
	Min / Max substrate resolution	2 / 3
Evolutionary Search	Algorithm *	CVT-MAP-Elites (fitness based replacements, random selection)
	Population size *	128
	Behavior Descriptor	Trajectory* + morphology
	Archive Size *	1000
Computational Setup	CPU	4 x Intel XeonCPU E5-2650
	Total number of cores	32

Table 5.2: Experimental setup of the "Extrinsic Guidance Experiment". All changes in relation to the "Towards Walking Creatures" experiment discussed in Section 5.2 are marked with (*).

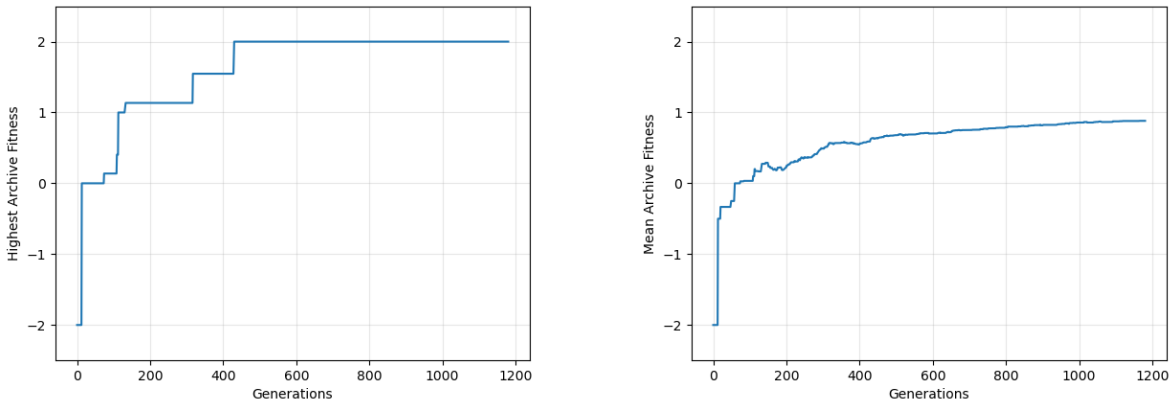
Results and Discussion

Given that the major insights gained from the failed experiments were already discussed above, this paragraph focuses on the results of the more successful (although still flawed) CVT-MAP-Elites experiment.

Figure 5.16 shows both the highest and the average fitness of creatures stored within the CVT-MAP-Elites archive. As discussed above, each creature initially starts with a fitness score of -2, which is increased to -1 if it has a valid morphology and again increased to 0 if it also has a valid controller. Figure 5.16a indicates that the first completely valid creature was evolved at generation 13. Given the population size of 128, this comes down to 1664 evaluations. In terms of wall clock time on the 32 core setup described above, this comes down to a mere 13 seconds. Given that all initially evaluated genomes are randomly generated, most of them are quickly discarded because they do not represent a valid morphology, which explains the rather high number of evaluations within the given time frame of 13

seconds before the first completely valid creature is evolved.

The first (and only) creature capable of reaching two target locations was evolved at gen-



(a) (b)

Figure 5.16: The highest (a) and mean (b) fitness of individuals stored in the CVT-MAP-Elites archive. Fitness is defined as the amount of target locations a creature was able to reach during its (bounded) lifetime, added to the inverse of the remaining distance to the latest goal (that the creature was not able to get to). Each creature initially has a fitness score of -2, which is increased to -1 if it has a valid morphology and to 0 if it has a valid controller as well. The first valid creature was evolved at generation 13 (evaluation 1664). Afterwards, a creature capable of reaching two target locations was evolved at generation 430 (evaluation 55040).

eration 430 (evaluation 55040). The wall clock time measured was approximately 4 hours and 55 minutes on the 32 core setup described above. This creature's morphology is shown in Figure 5.17. While we hoped for a higher amount of significant creatures, given the current strategy of spawning target locations (using a minimum and maximum difference between subsequent target directions) the task is somewhat challenging. Additionally, there are flaws[13] within the environment and experimental setup as discussed above. Nevertheless, the evolved creature succeeded at all four of the manual evaluations carried out, as shown

13. Besides the badly designed behavior descriptor, the randomness of the target locations leads to a noisy fitness function. Therefore it would have been better to evaluate each creature twice during optimization, with the resulting fitness of a creature being equal to an aggregation (e.g. the minimum) of the fitness scores of both evaluations.

in Figure 5.18. This was considered enough to conclude this experiment, given that the experiment was mainly meant as an initial validation of the controller to make sure that it is able to steer its morphology based on sensory-like observations that its given.

Given the rather erroneous setup of this experiment, further analysis (for instance concerning the evolved controllers and morphologies of the creatures) is held off until the experiment discussed in Section 5.3.3. Given that the insights gained within this experiments allowed to circumvent the same mistakes in that experiment, a more detailed analysis of the results will be given there.

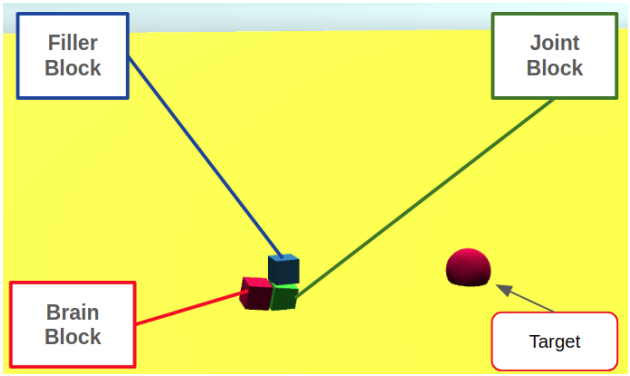
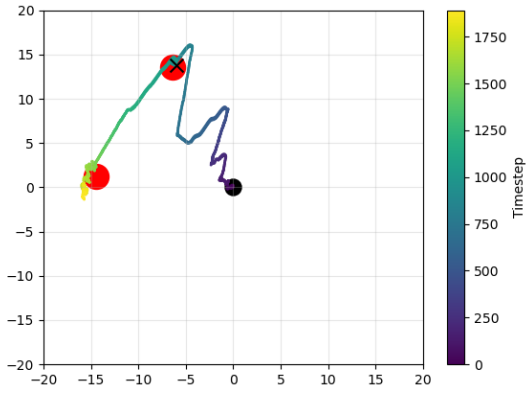
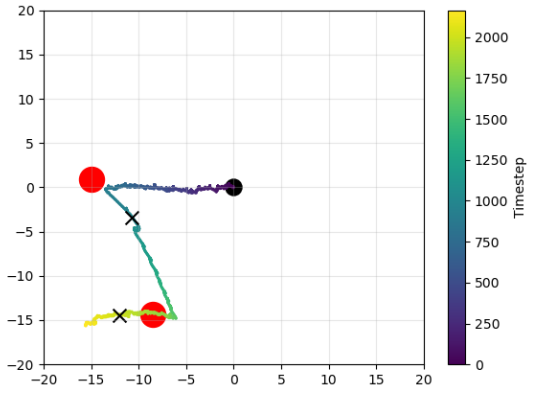


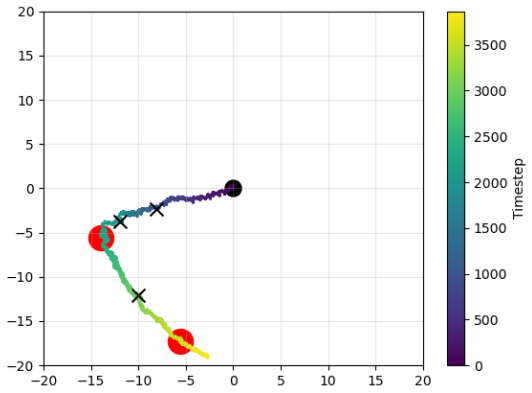
Figure 5.17: A creature within the three-dimensional environment, shown with morphology annotations and its current target location. This creature is the only one evolved that was able to reach two target locations.



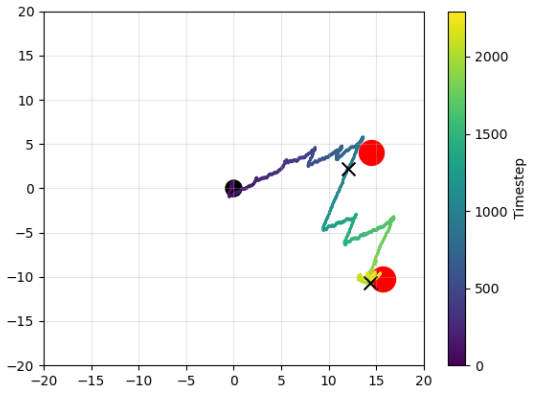
(a) Evaluation one



(b) Evaluation two



(c) Evaluation three



(d) Evaluation four

Figure 5.18: Visualizations of four evaluations of the creature that was capable of reaching up to two target locations. The creature's starting location is depicted as a black dot, while its subsequent target locations are shown in red. Its trajectory is shown with a color gradient denoting its location at every time step. Additionally, black crosses indicate the creature's location every 1000 steps, thereby giving a notion of speed. The creature was able to successfully reach the two target locations in all four evaluations. The jagged parts of some trajectories indicate that the creature cannot move in a straight line in every direction.

5.3.2 An Informative Detour of Failures

The previous experiment validated the ability of the controllers to process sensory-like observations in such a way that the controller is able to react to these observations by appropriately steering the morphology. This was done by manually providing the controller with the direction and distance toward a target location, to which it had to move the morphology. In other words, we gave it *extrinsic* guidance. More generally, this experiment served as an additional validation of the complete OCRA approach as well. Given that next to the controller, functional morphologies had to be simultaneously evolved, flexible enough to be steered to multiple randomly generated directions.

The next series of experiments aimed at introducing the previously unprovided sensor blocks to the creatures. In other words, to let the creatures evolve an appropriate placement for their sensors by themselves. Consequently, allowing the creatures to locate the target goals themselves and making the shift from extrinsic to *intrinsic* guidance. The type of sensor block that was used is described below, after which two experiments are discussed. While both of these experiments tried to evolve *"seeing"* creatures using different environments, neither of them succeeded. Nevertheless, their failure led to some useful insights concerning the OCRA methodology in general and the design of better environments. Therefore, these experiments are still briefly discussed.

Sensor blocks The type of sensor block implemented here casts a (user-defined) number of rays into the environment, which all report back if they have collided with an object and if so with which type of object and at what distance that object located. In this way, the sensor block can provide the creature with observations of both the environment (e.g. a target location sphere) and its own morphology (if a sensor block's ray collides with the creature itself for example). All rays are sent out of the same face of the sensor block. Figure 5.19

illustrates this.

In the same way as other types of available building blocks, these sensor blocks are incorporated by allowing the CPPN genomes to output a sensor block specific pattern (cfr. Chapter 3). This pattern denotes the presence of the sensor blocks within the *Agent Space*. Within each such a sensor block, a number of *input neurons* are placed, which allow to feed the actual observations made by that sensor block to the controller. However, the decision of which of these input neurons are connected to the controller is still made by the the *AESHN* neuroevolution algorithm (cfr. Section 3.2), which therefore uses the pattern of connection weights produced by the CPPN.

Next to a location (of a subcube) within the Agent Space, each sensor block of this type also requires a direction in which its rays are cast. Therefore, the CPPN genome was in this case initially augmented by one additional output that denotes this direction (one of the 6 faces of the sensor block cube, each corresponding to the direction of a positive or negative x-, y- or z-axis). This allows the CPPN genome to produce a specific pattern for the sensor blocks' directions. Given the -1 to 1 range of each CPPN output, we chose to interpret the value of this output pattern (queried at the sensor block location) by partitioning this output range as follows: $[0, 0.33]$ as the positive y-axis direction, $[0.33, 0.66]$ as the positive x-axis direction, $[0.66, 1.0]$ as the positive z-axis direction and the negations of all these ranges corresponding to the same, although negative axes.

The Maze Environment

The first experiments including these sensor blocks took place in a maze-like environment, shown in Figure 5.20. Within this environment, the creatures are guided through the maze by red barriers, which disappear upon touch. Performance (and fitness) here is defined by

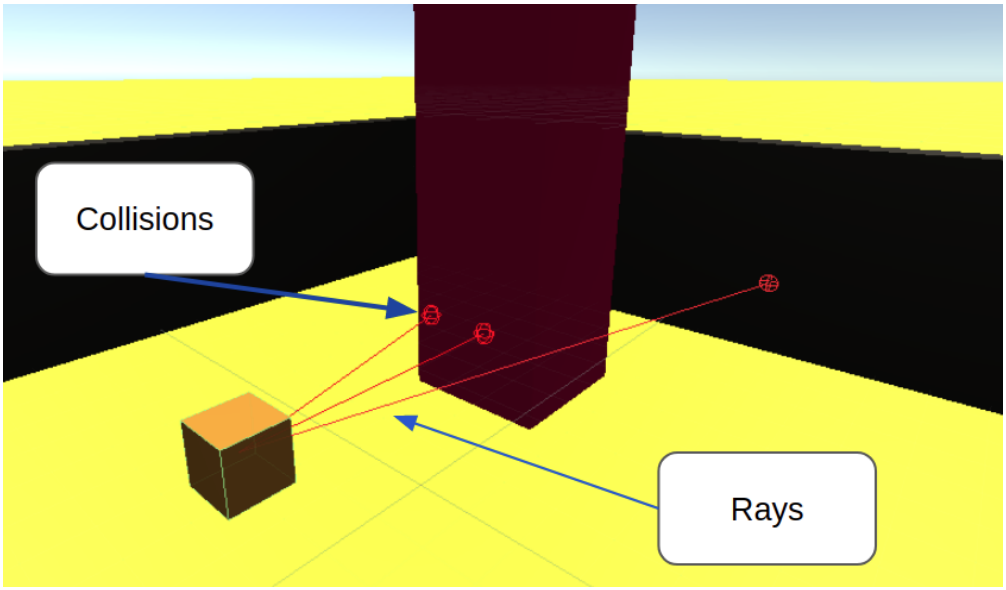


Figure 5.19: An example of a sensor block. This sensor block casts out three rays into the environment. These rays report back if they collided with an object and if so, the type of the object and the distance to that object. These observations are then fed to the *input neurons* located within the sensor block.

the amount of red barriers that a creature touched. The main idea behind these red barriers was to stimulate the creatures towards evolving sensor blocks (and appropriately using them), as they provide beneficial information to reach the end of the maze.

The experiments however resulted into creatures that were only capable of completing half of the maze, as indicated in Figure 5.20. Although all of these creatures incorporated sensor blocks within their morphology, all of them either had their sensor blocks pointing to useless directions (i.e. upwards or downwards) or did not connect the corresponding input neurons to the controller. This means that they were never able to actually "see" the red barriers, but just completed half of the maze by running into the wall and following it.

The issue concerning the impractical directions of sensor blocks was mitigated by redesigning the CPPN architecture. Instead of augmenting the CPPN with one additional output node, we augmented it with one additional output node per possible axis direction, i.e. we

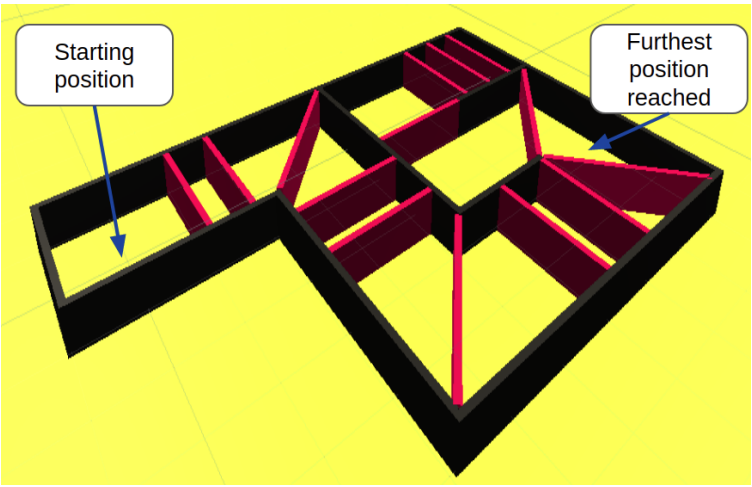


Figure 5.20: The Maze environment, made in Unity [113]. Within this environments, the creates have to follow the red barriers, which guide them towards the end of the maze. These red barriers disappear upon touch, thereby allowing the creature to move to the next barrier. The main idea behind these red barriers is to stimulate the creatures towards evolving sensor blocks and appropriately using them, given that they provide a huge advantage to quickly reach the end of the maze. The starting position and furthest position reached are both indicated.

unwrapped that single output node into three output nodes corresponding to the x-, y- and z-axis. The axis direction in which a sensor block casts its ray is then chosen based on which of these output nodes has the highest absolute activation. Based on the sign of the output activation, the direction (positive or negative) on that axis is then chosen.

This modification was tested in an additional experiment. This experiment resulted into creatures evolving useful sensor directions (i.e. pointing forward) and this time connecting the corresponding input neurons to the controller as well. Although the creatures still were not able to go past half of the maze, this did result into more independent runs (each of these experiments were run 8 times) evolving creatures capable of reaching half of the maze. Additionally, those creatures were evolved faster (less generations were necessary). This indicates that they now started to evolve towards actually making use of their sensors.

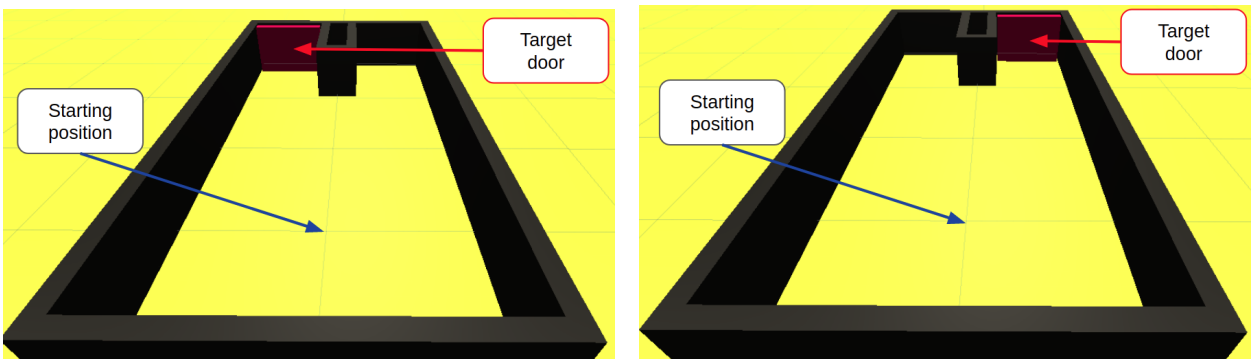
Next to the insights concerning the usage of the CPPN to extract sensor block directions, this series of experiments also indicated that this environment is not completely appropriate for the goal we want to achieve. Given that the goal here is to validate the ability of the OCRA methodology to evolve creatures capable of using their sensors, the fact that this environment allowed creatures that completely ignore this goal (e.g. by not really using their sensors) to still get decent performance scores renders it inadequate. When designing an environment, we thus have to be more cautious for insignificant solutions scoring well, thereby making sure that the environment truly stimulates the evolution towards what we are actually interested in. On a final note, the length of this maze environment also disadvantaged the evolutionary optimization. Given that many creature evaluations are necessary to explore the vast search space of solutions, it is beneficial that these evaluations are as short as possible.

The Monty Hall Environment

Taking into account the lessons learned from the Maze experiment above, we created a new environment that better fits the current goal of evolving "seeing" creatures. This environment is shown in Figure 5.21. Within this environment, the goal is to evolve creatures that walk to the red "door" during two consecutive trials. In each trial, the location of the target door is changed. Consequently, to solve this environment the creatures must evolve (and appropriately use) sensors.

The fitness of a creature during such a trial was set to 1 if it reached the target door, otherwise it was set to the inverse of the remaining distance towards the target door (to stimulate the evolution of moving creatures). The total fitness of a creature is then set as the minimum of its fitness scores of the two trials. This rather strict aggregation was used to force the usage of sensors.

Somewhat to our surprise, one of the eight independent runs of this experiment evolved a creature that seemed capable of solving this environment. However, after some analysis it quickly became clear that there was still an issue with the environment, given that the input neurons of the creature's sensor blocks were not connected. Instead, its controller had evolved a plethora of recurrent neurons which enabled it to extend the range of the periodic trajectory it makes the morphology follow. Given that we did not take the (memory) state of the controller into account when proceeding to the next trial, this allowed the creature to continue down that trajectory, resulting into reaching both doors. Figure 5.22 further illustrates this. A schematic visualization of this creature is shown in Figure 5.23. In a sense, it is fair to say that evolution somewhat outsmarted us here. However, after mitigating this issue by resetting the creature's controller in between trials, the creature was no longer able to reach both doors. Furthermore, none of the later experiments were able to evolve creatures that reached both doors.



(a) Trial 1 (b) Trial 2

Figure 5.21: The Monty Hall environment, made in Unity [113]. Within this environment, each creature is evaluated during two consecutive trials in which it must reach the target door (red). In each trial, the location of the target door is changed. Consequently, to solve this environment the creatures must evolve and make appropriate use of sensor blocks that allow them to "see" the target door.

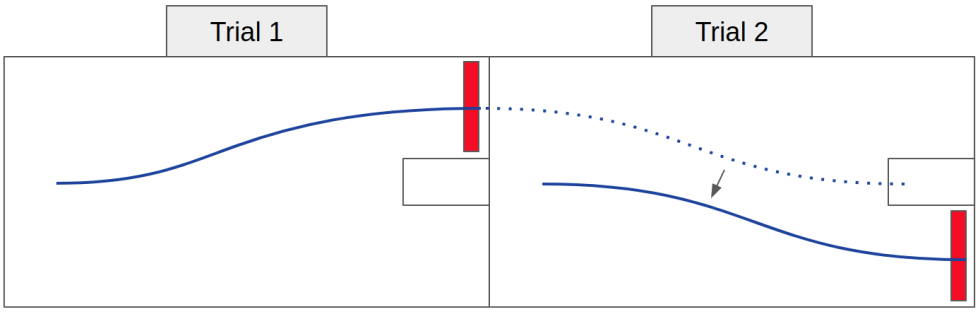


Figure 5.22: Illustration of how the extended periodic trajectory of the creature allowed it to reach both doors in the two consecutive trials of the Monty Hall environment. The creature's recurrent nodes allowed to extend the range of this periodic trajectory, such that when proceeding to the next trial it could just continue its trajectory in order to reach the next door. This was possible because the memory state was not reset when proceeding to the next trial.

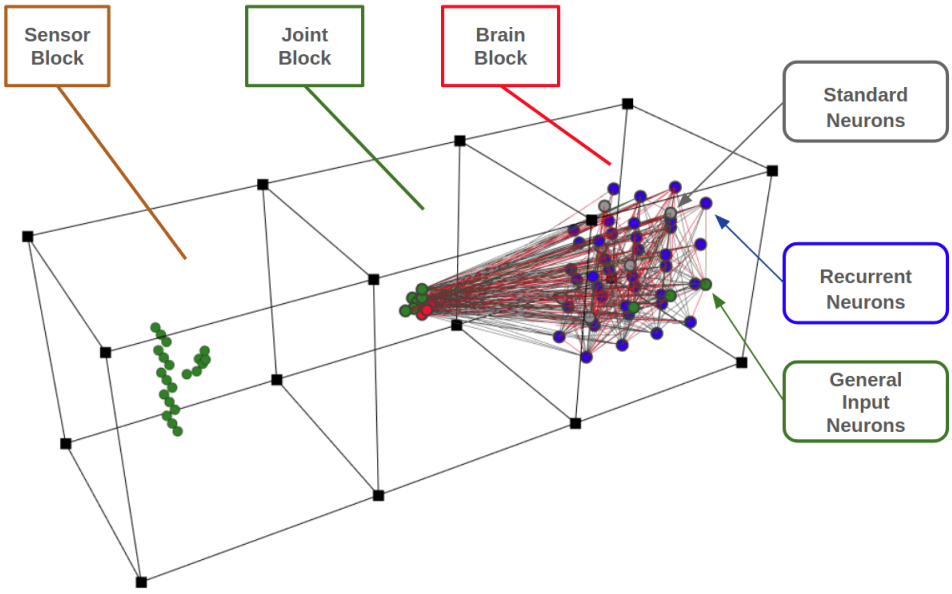


Figure 5.23: Schematic visualization of the snake-like creature that reached both doors in the two consecutive trials of the Monte Hall environment. As can be seen, none of the input neurons corresponding to the sensor block were connected. The controller within the brain block contains 7 standard neurons (grey), 39 recurrent neurons (blue) and 671 connections in total. While positive connections are colored black, negative connections are colored red.

5.3.3 Reducing Complexity through Crawlers

The failure of the Monte Hall Experiment indicated a need for a complexity reduction.

On the one hand, we introduced an even simpler environment, named *"Red Wall"* (discussed in the paragraph below). On the other hand, we simplified the creatures themselves as well. Instead of letting them evolve an appropriate placement and amount of sensor blocks, we fixed four sensors to the brain block (covering each horizontal direction). This is illustrated in Figure 5.24. The input neurons corresponding to those sensors are presented to the brain through general input nodes (cfr. Section 3.2). This is still different from the extrinsic guidance experiment described in Section 5.3.1, in the way that the observations are now provided through the ray based sensor mechanics described above, in contrast to directly providing them with the direction and distance to the target. This allows us to more specifically validate the ability of the controller to handle this type of sensors.

To make sure that the sensors would not get hindered by other building blocks of the creature, we also restricted the Agent Space such that the OCRA algorithm can now only place blocks beneath the brain block. Additionally, this reduced the morphological complexity, given that less building blocks can be placed. To differentiate this more constrained type of virtual creature from the more freely specified ones above, we named them virtual *crawlers*.

The Red Wall Environment

Somewhat similar to the Monte Hall environment, the goal is again to evolve creatures (or in this case, crawlers) that move to the red target during two consecutive trials. One creature evaluation is thus composed out of two trials. The environment is shown in Figure 5.25. In each trial, the location of the target is changed. Consequently, to solve this environment the

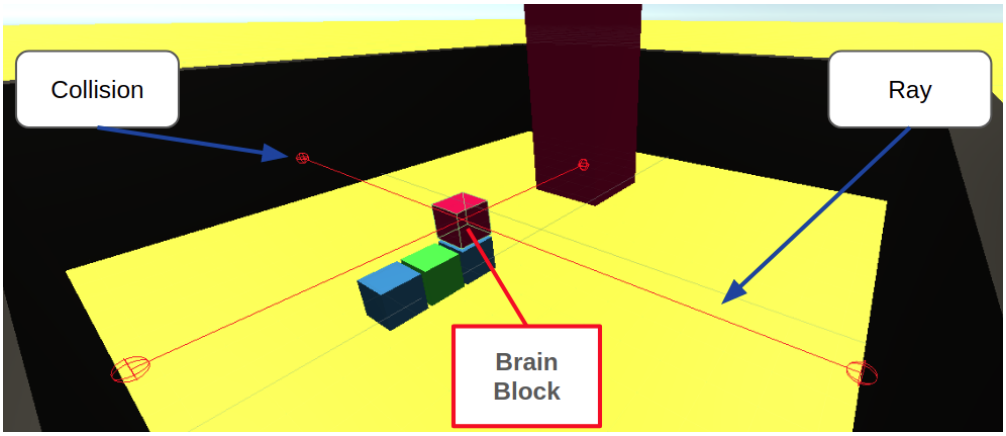


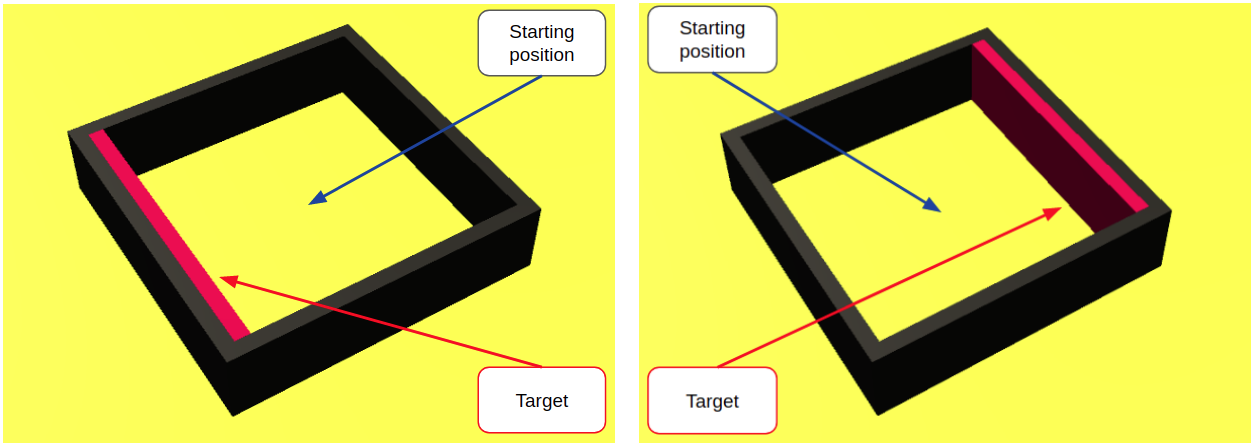
Figure 5.24: An example of the sensor incorporation within the brain block. Four sensors, that each cast out one ray, provide information to the crawler concerning its environment.

crawlers must evolve such that they appropriately use their the sensor observations (with the sensors being fixed to the brain block). The main difference with the Monte Hall environment is that the distance from the starting position to the target is now a lot smaller, allowing for faster evaluations. Additionally, there is now a more distinct difference between the paths to the target of each trial.

The crawlers are given 2000 time steps to complete each trial. In between such trials, their controller is now reset to prevent the same memory related issue from happening as encountered in the Monty Hall environment. The fitness of the crawler in each trial is set to 1 if it reached the target, with an additional factor that increases the fitness based on the number of remaining time steps (thereby encouraging faster crawlers). If the crawler was not able to reach the target within a trial, its fitness for that trial is calculated based on how much closer it got to the target compared to its starting location (0 being equal to not having gotten closer and 0.99 to nearly touching it). The total fitness of a crawler is then set as the minimum of its fitness scores of its two trials. Again, this rather strict aggregation is used to force the usage of sensors and preventing insignificant solutions to receive a high score. Additionally, we again incorporate the validity of the crawlers into their fitness score

by initially setting it to -2, increasing it to -1 if it has a valid morphology and to 0 if it has also has a valid controller.

Given that there is no inherent randomness within this environment, one evaluation consisting out of two trials (each with a fixed but different target location) suffices to truly evaluate the performance of a crawler. Furthermore, granted that the crawler's controller is reset in between the trials, we prohibit an advantageous information leak that could allow the crawler to "memorise" the path towards the correct target in the second trial based on its path in the first trial. In other words, we do not have to evaluate a crawler multiple times in order to be certain of its performance.



(a) Trial 1 (b) Trial 2

Figure 5.25: The Red Wall environment, made in Unity [113]. Within this environment, each creature is evaluated during two consecutive trials in which it must reach the target wall (red). As shown, the location of the target wall is different within each of these two trials, although fixed per trial. Consequently, to solve this environment the creatures must evolve towards making appropriate use the sensor observations that they are given.

Experimental Setup

CVT-MAP-Elites was used as the underlying framework to guide the evolutionary process, with the same behavior descriptor[14] as discussed in the *Extrinsic Guidance Experiment* above. The number of archive cells was however reduced from 1000 to 100, thereby increasing selection pressure and putting more focus on the *Quality* component of the *Quality-Diversity* trade-off.

The mutation and crossover operators defined in the NEAT algorithm (cfr. Section 2.5.2) were again used to introduce variation in the CPPN genomes. The CPPN mutation probabilities were set to 0.03 and 0.005 for connection additions and removals respectively, 0.02 and 0.005 for node additions and removals respectively and 0.94 for connection weight updates.

Although not previously mentioned the AESHN neuroevolution algorithm's variance thresholds were set to 0.3 for the division phase, 0.03 for the pruning phase and 0.3 for the band pruning phase. These are lower than the ones used in the previous experiments to allow more complex controllers to arise, given that they have to deal with the additional complexity of processing the sensor observations. Furthermore, the maximum substrate resolution was increased from 3 to 4. This corresponds to the maximum depth of the quadtree representation of the hypercube (cfr. Section 2.5.4), leading to a maximum neural complexity of 4096 neurons within the three-dimensional *Brain Substrate*. However, it is important to note that the chance of reaching such a neural complexity is rather low given the variance thresholds; this mainly allows the AESHN algorithm to increase density in the regions of the artificial brain where it is deemed necessary.

14. While this was a bad choice for the "Extrinsic Guidance Experiment", the issues encountered there are not reflected within this environment (due to the target locations being fixed in each trial).

A schematic visualization of the placement of the *general input neurons* within the brain block of a simple crawler is presented in Figure 5.26. Next to the bias, sine and cosine wave inputs, the input neurons corresponding to the four sensors are now included as well. Each of these requires six input neurons per ray that it casts, given that each ray returns the following list of observations: 0 or 1 indicating if the ray hit an object, the normalized distance travelled by the ray until collision (1 meaning no collision) and a one-hot encoding indicating the type of object that was hit. The one-hot encoding corresponds to 4 input neurons given that we have four object types here: (1) the crawler itself, (2) the ground plane, (3) the wall and (4) the target wall. In this experiment, the number of rays per sensor was set to 1.

To increase the soundness of the results, we conducted eight independent runs of this experimental setup. Each such run was done on the same 32 core compute node as in the experiments before, using multiprocessing to take maximal advantage of the four *Intel(R) Xeon(R) CPU E5-2650 v2 octa-core @ 2.60GHz* CPUs available. Here as well, we chose to make the population size dependent on the number of cores available as a way to make sure that every core was maximally taken advantage of. This resulted into a population size of 128 or four times the number of cores available. Table 5.3 summarizes the experimental setup used for the results discussed below.

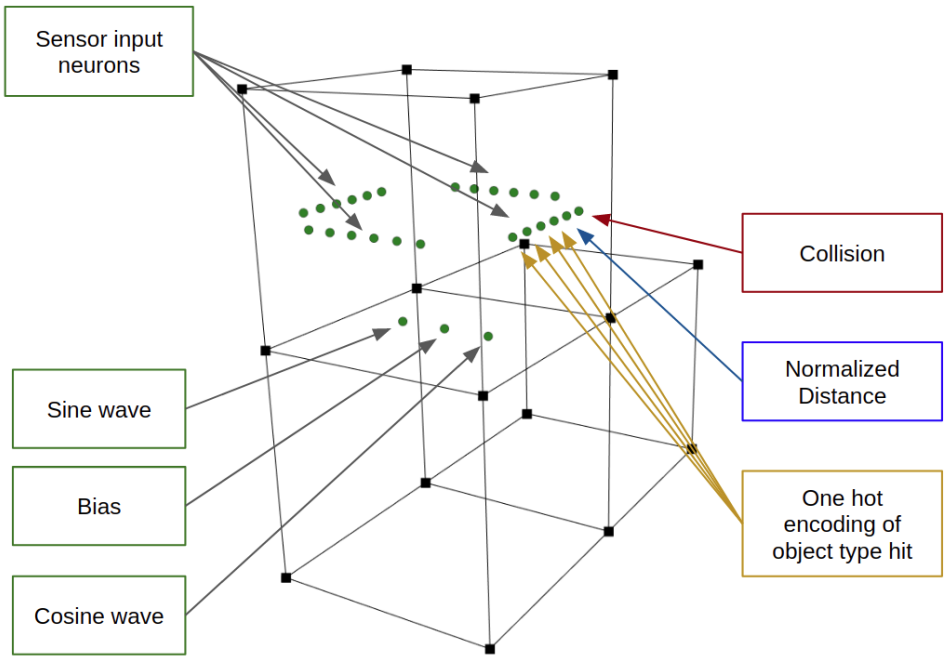


Figure 5.26: Schematic visualization of the placement of the *general input neurons* within the brain block of a simple crawler. These general input neurons are placed in the outer region of the brain block, as the inner region is reserved for the hidden neurons placed by the AESHN algorithm. The general input neurons corresponding to the bias, sine and cosine wave inputs are placed at the bottom. The general input neurons corresponding to the four sensors are placed at their respective face of the brain block, from which they cast their ray outwards. Each of these sensors are coupled with six input neurons per ray that they cast (in this case one ray per sensor), given that each ray returns six observations as shown here.

CPPN mutation probabilities	Neuron addition / removal *	0.02 / 0.005
	Connection addition / removal *	0.03 / 0.005
	Connection weight update	0.94
AESHN	Division phase VT *	0.3
	Pruning phase VT *	0.03
	Band-pruning phase VT *	0.3
	Min / Max substrate resolution *	2 / 4
Evolutionary Search	Algorithm	CVT-MAP-Elites (fitness based replacements, random selection)
	Population size	128
	Behavior Descriptor	Trajectory + morphology
	Archive Size *	100
Computational Setup	CPU	4 x Intel XeonCPU E5-2650
	Total number of cores	32

Table 5.3: Experimental setup of the "Red Wall Crawlers" experiment. All changes in relation to the "Extrinsic Guidance Experiment" discussed in Section 5.3.1 are marked with (*).

Results and Discussion

Three out of the eight runs we conducted were able to evolve crawlers capable of reaching the target in both trials. While two of these three successful runs maintained a similar low level of phenotypic controller complexity, one run deviated and evolved a much higher level of complexity. Due to this difference, this run will be analysed separately from the others at the end of this section.

Figure 5.27 gives an overview of the highest fitness of an individual stored in the CVT-MAP-Elites archive for each of those seven other runs[15]. The fitness of an individual here is set as the minimum of its fitness achieved in both its trials. The fitness in each trial is set to 1 if it reached the target, with an additional remainder that increases the fitness based on the number of remaining time steps the crawler had to reach the target (thereby encouraging faster crawlers). Additionally, analogous to the previous experiments, each creature initially starts with a fitness score of -2, which is increased to -1 if it has a valid morphology and again increased to 0 if it also has a valid controller.

As can be seen in the figure, run 2 and run 7 both evolved crawlers that reached both targets (fitness higher than 1). While run 7 already did so after 54 generations (equivalent to 6912 evaluations given the population size of 128), run 2 took a little while longer and did so after 394 generations (equivalent to 50432 evaluations). In terms of wall clock time on the 32 core setup described above, run 7 evolved its first crawler that reached both targets after 22 minutes and run 2 after 14 hours and 54 minutes. Both of these runs then further improved upon these initial crawlers by decreasing the number of time steps required to reach both targets. In this way, the best crawler over all runs had a fitness score of 1.634. This means that it completed both its trials successfully, with its "worst" trial only requiring 36% of the 2000 time steps given.

Given that there is no inherent randomness within this environment, one evaluation consisting out of two trials (each with a fixed but different target location) suffices to truly evaluate the ability of a crawler to appropriately use its sensor observations. This means that it is impossible for crawlers to blindly reach the target in both trials and thereby diminishing the validity of these results.

15. Important to note is that this is not equal to the highest fitness measured within the set of evaluated crawlers of each generation.

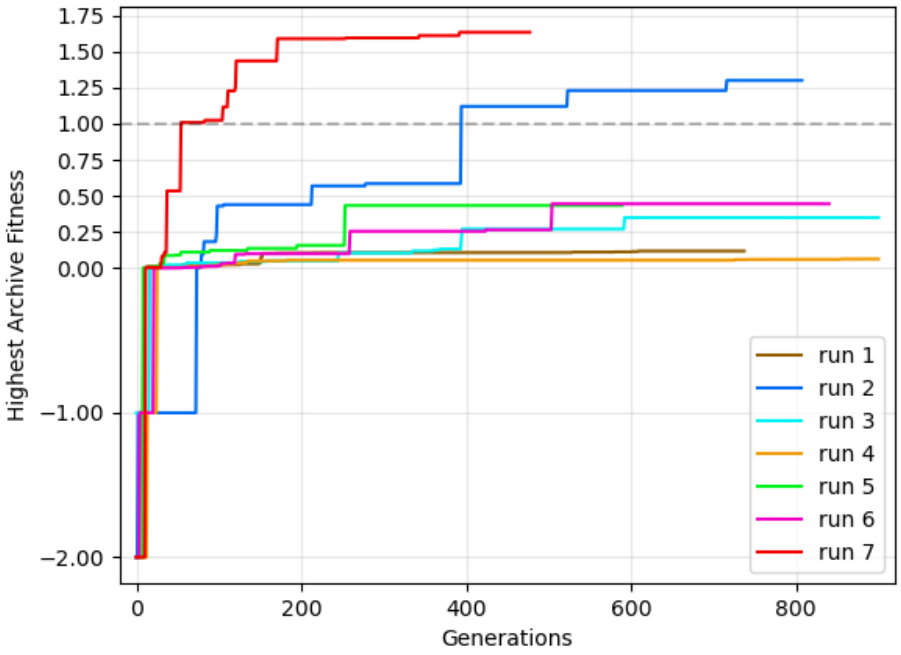


Figure 5.27: Highest fitness of an individual stored in the CVT-MAP-Elites archive, shown for 7 independent runs. Some runs are shorter than others due to compute node crashes. As can be seen, run 2 and 7 evolved crawlers capable of reaching their target in both trials (fitness ≥ 1). Moreover, these runs then evolved even better crawlers that more quickly completed both trials. While run 1 and 4 were not able to evolve any significant solutions, the other runs (3 , 5 and 6) evolved crawlers that did however move towards the right target in both trials.

Before diving further into the results of those best two runs (and the additional third run that also solved this environment), we will first take a look at the other less successful runs. Their highest fitness scores are also shown in Figure 5.27. While two of those other runs (1 and 4) were not able to evolve any significant solutions, the other runs were able to evolve crawlers that reached fitness scores between 0.25 and 0.50. Although the crawlers evolved by those runs did not reach their target goal in both trials, given that the fitness of the two trials is aggregated by the minimum operator, the crawlers must have moved towards the target in each trial. This indicates the possibility these runs would also have evolved crawlers reaching both targets if they were given more generations.

While it is not surprising that not all runs led to equally successful results (evolution remains a stochastic process), we can look for a possible explanation by taking a look at the differences in both genotypic and phenotypic (controller) complexity between the two best runs (2 and 7) and the two worst runs (1 and 4). This is shown in Figure 5.28. As can be seen, the better performing runs have evolved a higher average amount of hidden neurons (Figure 5.28a), but a lower average amount of connections within their CPPN genomes (Figure 5.28b). This suggests the reason why run 1 and 4 were not able to evolve creatures that reach both targets; there either was a problematic lack of structural mutations to their genomes (which is plausible given the rather low mutation probabilities discussed in the experimental setup above) or there were too many connection related structural mutations compared to neuron related structural mutations. Given that we allow maximally one structural modification to occur per mutation (we do not want to change the CPPN too much in one mutation, and structural mutations are quite intense), it is possible that the ratio of connection based to neuron based mutations was unbalanced in these inferior runs. This, combined with our rather strict CPPN query usage (we never use all input and

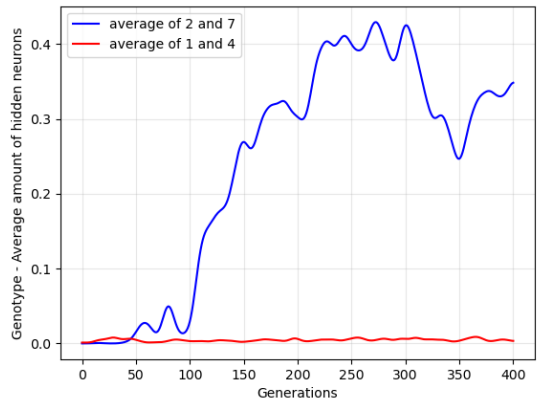
output neurons at the same time, cfr. Section 3.3) which mitigates the need for many connections, clarifies why these runs may have performed worse: they did not evolve towards functional genomes and consequently functional phenotypes. This is also in line with the decreasing amount of CPPN connections in the better performing runs, which in contrary did evolve such functional genomes. Furthermore, the consequences of this can also be seen in the differences in terms of phenotypic controller complexity (Figures 5.28c and 5.28d). Nevertheless, given that all of these mutations occur stochastically, this issue is a natural artifact of the evolutionary process.

While the two best performing runs were able to evolve creatures that successfully reached both target doors, it is still somewhat surprising that their average genotypic complexity remained rather low. Particularly, the average amount of evolved hidden neurons did not exceed 1. Although it is possible that the CPPN query usage employed in the OCRA approach (cfr. Figure 3.4 mitigates the need for a higher level of genotypic complexity, an equally reasonable argument might suggest that this is a consequence of the lack of an innovation protection mechanism[16] in the CVT-MAP-Elites framework. This forms the basis for the extension of the CVT-MAP-Elites framework proposed in Chapter 4 and to which we will come back to in Section 8.3 when discussing the virtual ecosystem experiments.

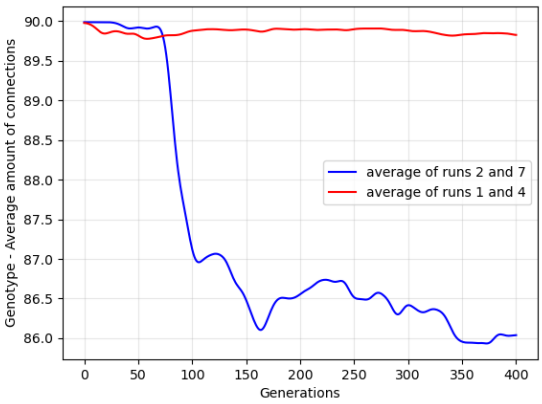
The best crawler[17] over all runs was evolved by run 7 and is shown in Figure 5.29. This crawler was evolved after 392 generations, which is equivalent to 50176 evaluations given the population size of 128. While Figure 5.29a gives a schematic overview of the whole

16. An innovation protection mechanism prevents recently structurally mutated genomes from being discarded due to an unfair comparison with older (and better tuned) genomes.

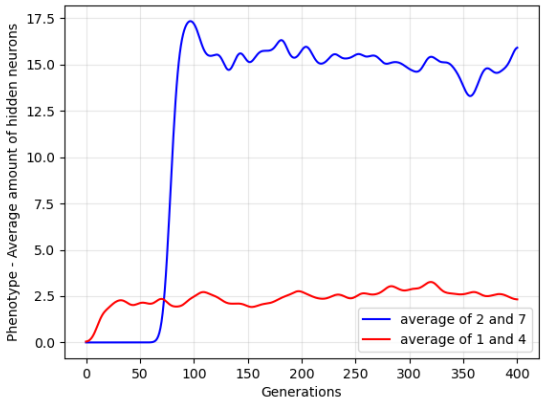
17. A video demonstrating its evaluation over the two consecutive trials is available at: `https://www.shorturl.at/dvR01`



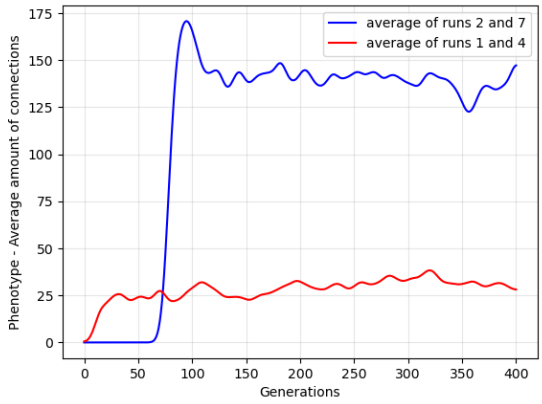
(a) Genotypic complexity - average amount of hidden neurons over time.



(b) Genotypic complexity - average amount of connections over time.



(c) Phenotypic controller complexity - average amount of hidden neurons over time.



(d) Phenotypic controller complexity - average amount of connections over time.

Figure 5.28: Visualizations of both average genotypic (CPPN) and phenotypic controller complexities depicted over time. Each plot indicates the difference between the average of the two best runs (2 and 7) and the two worst runs (1 and 4). As can be seen, the average amount of hidden neurons in the CPPN genomes of the better performing runs is higher than in the inferior runs (a). However, the average amount of connections in the better runs is lower than in the inferior runs. In terms of phenotypic controller complexity, both the average amount of hidden neurons and connections in the better performing runs is higher than in the inferior runs. The complexities are cut-off at generation 400 to improve visibility.

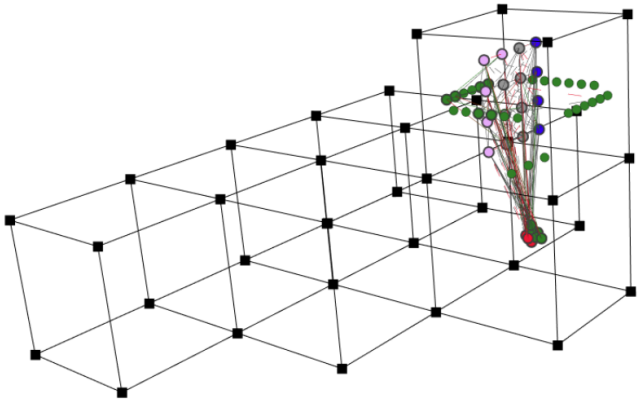
creature, Figure 5.29b gives a top-down view of its evolved brain. The interesting thing of this top-down view is that it shows which of the provided sensor related input neurons were attached to the controller. As can be seen, only a select few were connected. However, these were enough to locate (and move to) the target in both trials. Given that the targets of both trials are located at opposite sides of the square environment, knowing that the target is not on the one side is enough to know that it is on the other. This showcases the natural feature selection capacity of the employed AESHN neuroevolution algorithm.

Next to this crawler, run 7 also evolved 187 other crawlers that successfully completed the task. Three of these other successful crawlers are shown in Figure 5.30. Additionally, the morphological diversity over time of the evolved (successful) crawlers of run 7 is shown as a cumulative sum in Figure 5.30d. On the controller side, Figure 5.31 visualizes the proportions of the different hidden neuron types of the successful creatures of run 7 over time. This figure indicates that every successful crawler always incorporated all three types of hidden neurons within its controller, namely (1) standard neurons, (2) recurrent neurons and (3) neuromodulatory neurons[18].

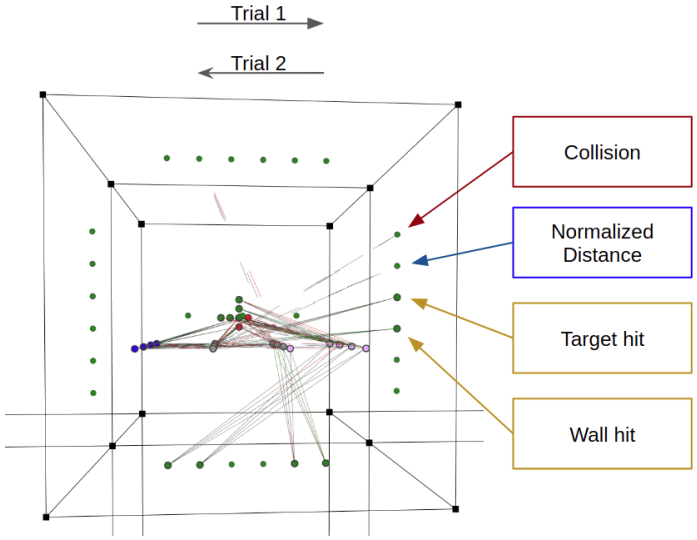
Next to run 7, run 2 also evolved 6 crawlers capable of reaching the target in both trials. All of these 6 crawlers had the same morphology and a schematic overview of such a crawler is shown in Figure 5.32. As can be seen in the figure, their morphology is quite similar to the previously shown morphologies evolved in run 7. However, on the controller side this run took a different path and did not evolve any neuromodulatory neurons. This in turn led to the crawlers moving in a different[19] (and somewhat more peculiar) way compared to the ones of run 7.

18. Neuromodulatory neurons change the connection weights of other (non-neuromodulatory) neurons depending on their own activation (cfr. Section 3.2).

19. A video demonstrating this movement is available at: `https://www.shorturl.at/yHMR4`

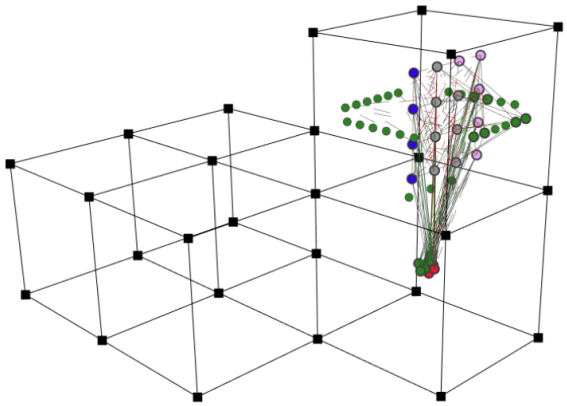


(a) Schematic visualization of the best performing crawler. This crawler's morphology is composed out of 7 filler blocks, 1 joint block and 1 brain block. The controller evolved within the brain block has 7 standard neurons (grey), 4 recurrent neurons (blue), 5 neuromodulatory neurons (pink) and 171 connections in total.

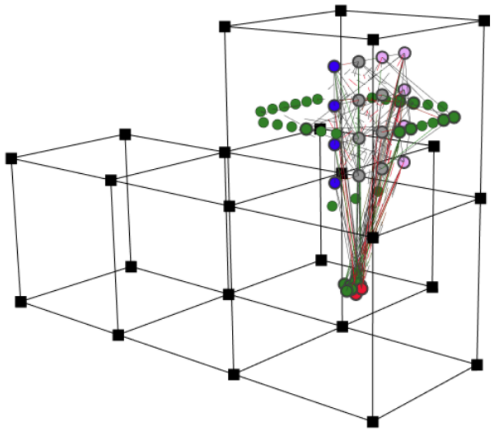


(b) A top-down view of the evolved brain of the best performing crawler. The most important connected (sensor related) input neurons are annotated.

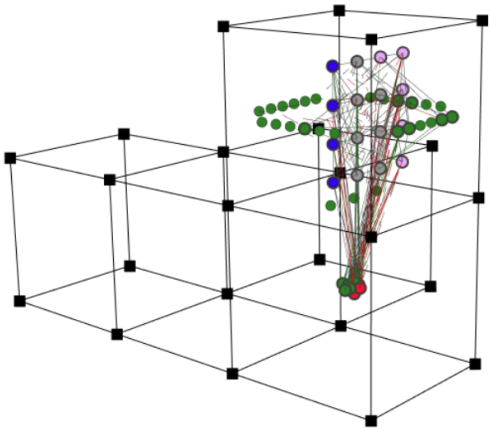
Figure 5.29: Schematic visualizations of the best performing crawler. This crawler was evolved by run 7 after 392 generations (equivalent to 50176 evaluations) and obtained a fitness of 1.634. In (a) the complete crawler is shown, while in (b) a top-down view of its evolved controller is shown. The top-down view shows which sensor related input neurons were connected. This view indicates that the crawler was able to reach the target in both trials by using the sensor which points to the target of the first trial.



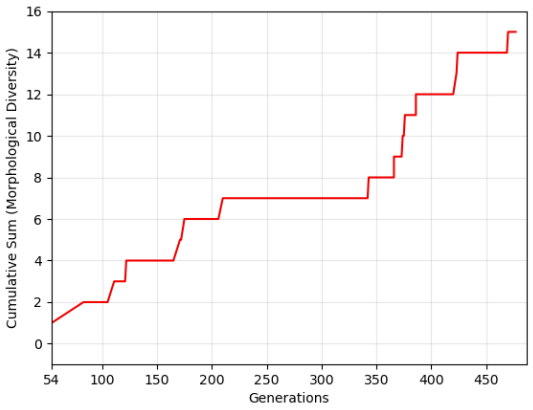
(a) Evolved in generation 54 with a fitness of 1.01. Its controller contains 7 standard neurons, 4 recurrent neurons, 5 neuromodulatory neurons and 174 connections.



(b) Evolved in generation 120 with a fitness of 1.25. Its controller contains 7 standard neurons, 4 recurrent neurons, 5 neuromodulatory neurons and 178 connections.



(c) Evolved in generation 453 with a fitness of 1.6. Its controller contains 8 standard neurons, 4 recurrent neurons, 4 neuromodulatory neurons and 176 connections.



(d) Cumulative sum of the amount of different morphologies of successful crawlers over time.

Figure 5.30

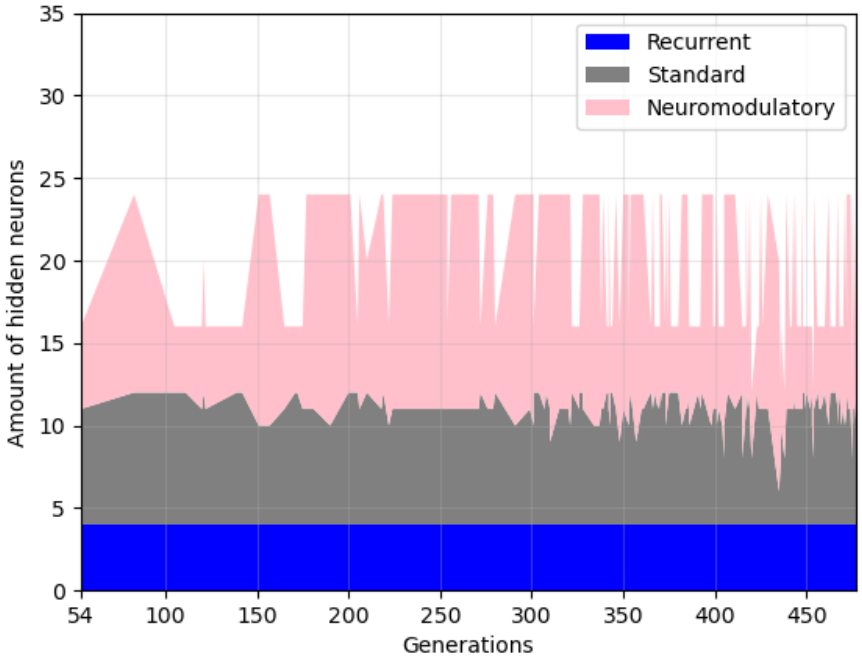


Figure 5.31: The proportions of the different types of hidden neurons used in the controllers of successful crawlers over time. As can be seen, all three types were always incorporated in the controllers. The amount of recurrent neurons (blue) always remained four, while the amount of standard (grey) and neuromodulatory neurons (pink) was less stable.

Lastly, we will take a look at the previously unshown results of the eighth run. As mentioned

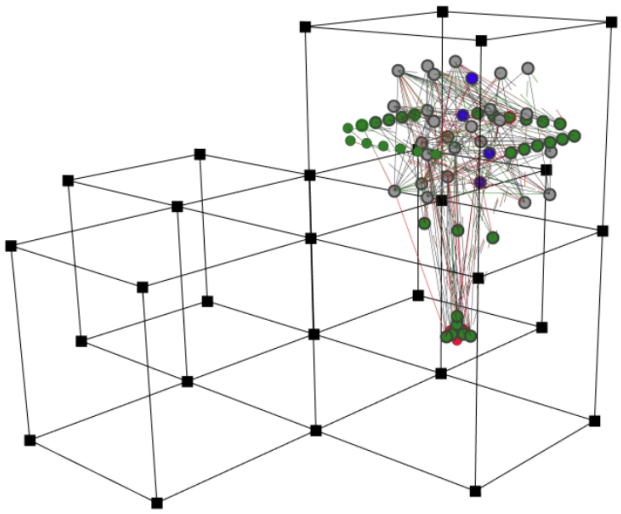
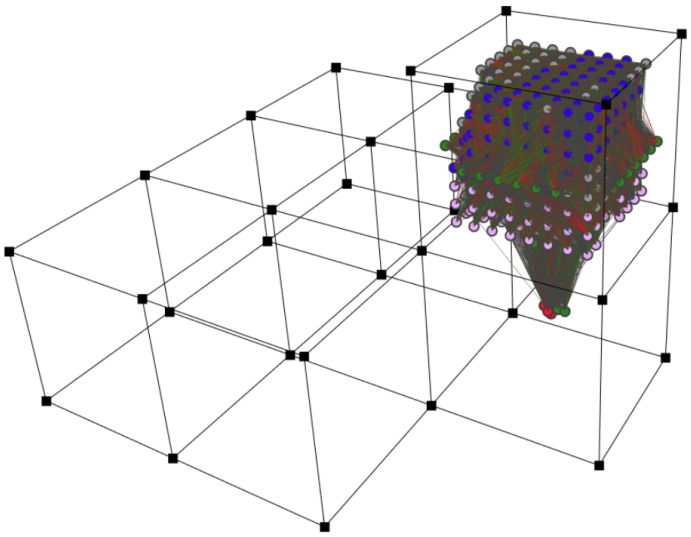


Figure 5.32: Schematic overview of a successful crawler evolved in run 2. While the morphology is quite similar to that of the crawlers evolved in run 7, the evolved controller differs more as no neuromodulatory neurons were incorporated into the controller. This crawler was evolved after 716 generations (equivalent to 91648 evaluations) and obtained a fitness of 1.3. Its controller contains 25 standard neurons (grey), 4 recurrent neurons (blue) and 256 connections in total.

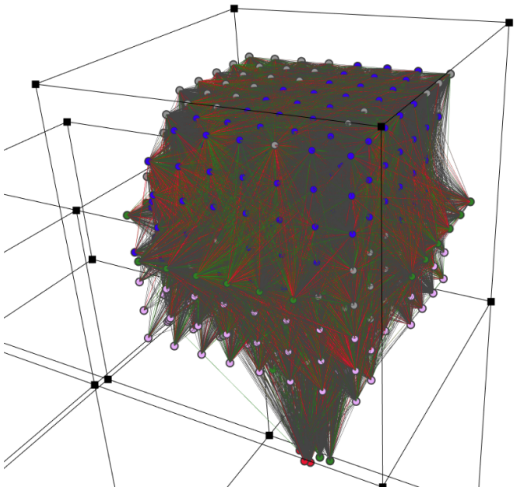
above, this run deviated from all other runs by evolving towards much higher phenotypic controller complexities of up to approximately 400 hidden neurons and 170000 connections. Although the phenotypic controllers were much more complex, the morphologies remained similar to the other runs. The first creature capable of reaching the target in both trials was evolved after 40 generations (equivalent to 5120 evaluations). While this is 14 generations less than run 7, in terms of wall clock time on the 32 core compute node it took 17 hours and 7 minutes longer. The reason for this is that the increase in controller complexity also increases the computational complexity of building (and evaluating) such crawlers.

A schematic overview of the best performing crawler of this eighth run is shown in Fig-

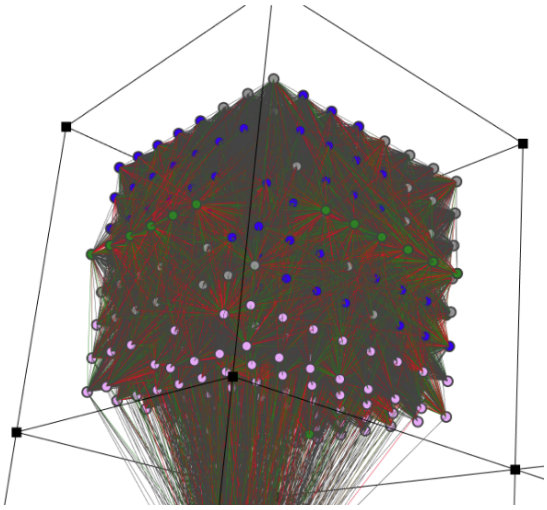
ure 5.33a. Figures 5.33b and 5.33c zoom in on its evolved controller and present it from multiple angles, thereby showcasing its modular structure. Intuitively, this modular structure consists of a memory module (recurrent neurons), a learning module (neuromodulatory neurons) and a processing module (standard neurons). While this structure is a rather interesting product of the evolutionary process, next to being visually appealing and somewhat resembling the modular structure of biological brains, the environment in which it was evolved is too simple to truly investigate its potential benefits. However, given that this crawler was already evolved after 42 generations, only a maximum of 42 mutations to its underlying CPPN genome could have occurred. Additionally, this CPPN did not have any hidden neurons. This suggests that its performance mainly originates from the powerful encoding capabilities of the underlying CPPN genome from which this modular structure is derived.



(a) The crawler's morphology is composed out of 6 filler blocks, 1 joint block and 1 brain block.



(b) Viewpoint one



(c) Viewpoint two

Figure 5.33: Schematic visualizations of the best performing crawler of run 8. This crawler was evolved after 42 generations and obtained a fitness of 1.362. (a) gives an overview of its morphology, while (b) and (c) zoom in on its evolved controller, thereby showcasing its modular structure. The controller contains 43 standard neurons (grey), 357 recurrent neurons (blue), 143 neuromodulatory neurons (pink) and 167409 connections in total. Positive connections are colored black while negative connections are colored red. Intuitively, this modular structure consists of a memory module (recurrent neurons), a learning module (neuromodulatory neurons) and a processing module (standard neurons).

Part II

The Virtual Ecosystem

CHAPTER 6

A VIRTUAL ECOSYSTEM IN UNITY

Besides introducing a novel methodology towards the simultaneous evolution of agent brain and morphology (Part I), the second aim of the research presented in this thesis is to take a next step towards unshackling the full potential of virtual evolution by producing a more comprehensive environment in which that evolution may take place. This chapter presents this environment, which was chosen to be a virtual ecosystem.

Section 6.1 starts by giving an overview of the virtual ecosystem. Next, Section 6.1.1 discusses the general flow of simulations within this virtual ecosystem. Lastly, Section 6.2 motivates our choice for Unity [113] as the simulator.

6.1 Overview

One of the first decisions that has to be made when commencing an investigation into artificial living systems is that of scale, i.e. at what level of detail is it desirable to specify the parameters and underlying models of the simulation, and at what level does one wish to observe the resultant behaviors [25]. Given our aim is to allow for ethological behaviors of embodied agents with a "higher-level" of intelligence, it makes sense to forgo the preceding sub-atomic physics, molecular and cellular levels. We thereby sidestep what has taken millions of years of evolution in our biological world and in our case modeling the virtual ecosystem and its inhabiting organisms at the level of "higher order building blocks".

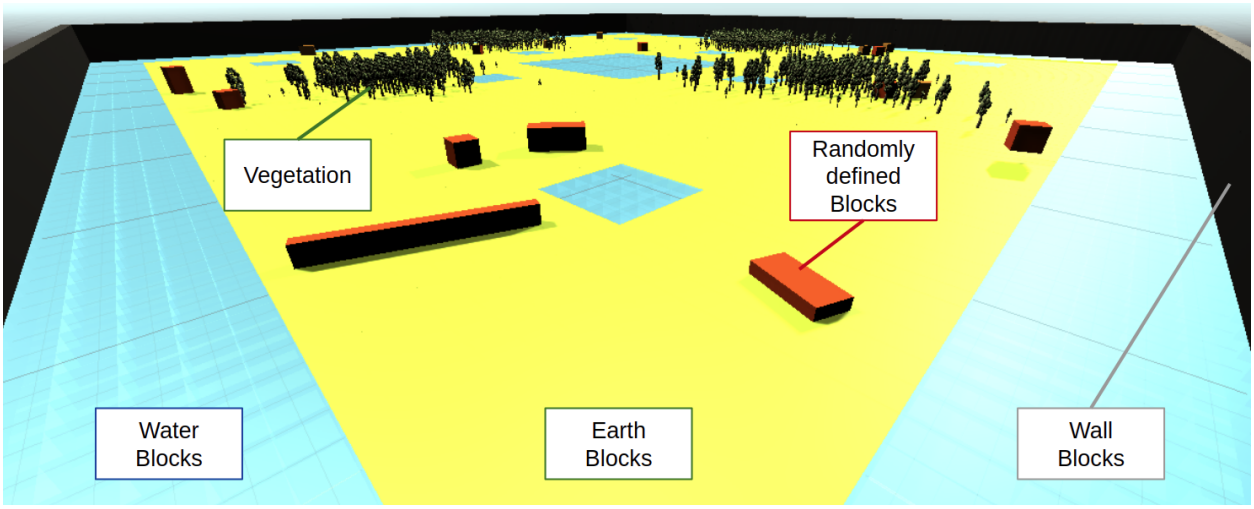
In terms of the virtual ecosystem, the building blocks used here include *earth*, *water*, *wall* and *randomly defined* blocks. Patches of earth blocks establish the land on which the artificial organisms live, while patches of water blocks form the "oceans" in between those patches

of land. Wall blocks can be used to create a visible boundary of the environment, next to inserting impassable barriers to isolate parts of the inhabiting population. Such impassable barriers are used in a related work named PolyWorld [25] as a way to stimulate the emergence of species (cfr. Section 2.2.2). The usage of the fourth type of block is related to the interactions between the inhabiting organisms and this environment and will be further discussed in Section 7.2.3. Figure 6.1 presents an example virtual ecosystem containing all block types.

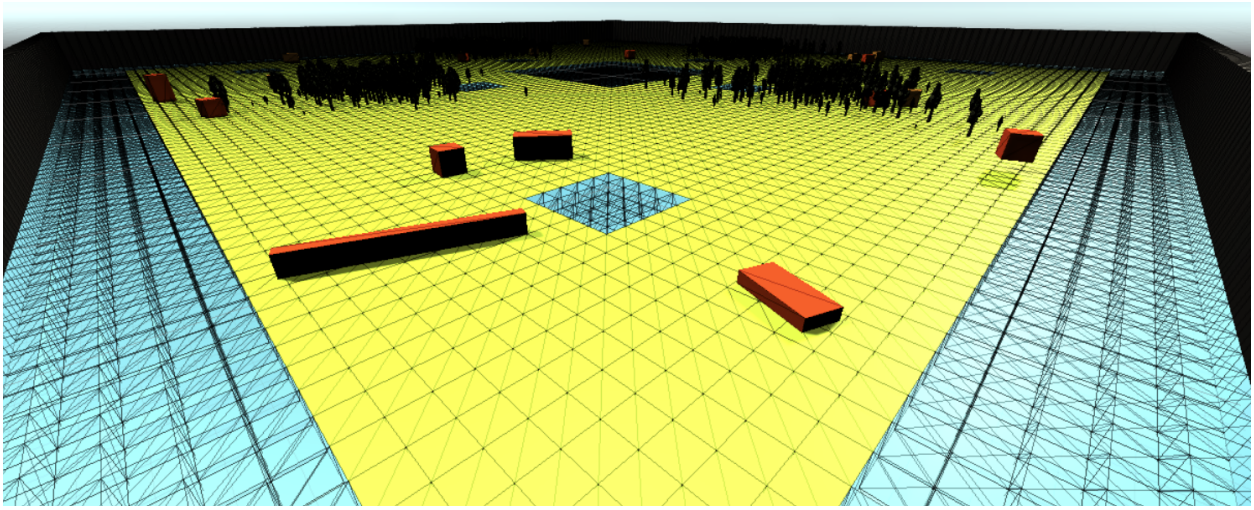
The important thing to understand here, is that the proposed virtual ecosystem has a modular structure and is in essence entirely composed out of building blocks. This approach is inspired by a popular game named *Minecraft* [117]. In this game, the players can explore a world composed out of rough three-dimensional objects - mainly cubes and fluids, and commonly called "blocks" - which represent various materials such as dirt, stone, ores, tree trunks, water and lava. Given its vast range of possibilities and open-ended nature, Minecraft has already been employed as the environment for research in both the reinforcement learning [118] and evolutionary computation [119] domains.

As discussed in Section 1.1, the key idea behind using such a modular world is that it allows to more easily introduce complex environmental dynamics, such as ever-changing landscapes due to blocks being moved around or changes being made to the properties of the inserted blocks (either stochastically or by the user). Additionally, such a bottom-up approach allows for a natural expansion of the range of possible interactions between the inhabiting organism and the environment. This is further discussed in Section 7.2.3.

The proposed virtual ecosystem hosts two types of organisms which both are subject to evolution, although at different levels of complexity. The first and most prominent type are the *virtual creatures* discussed in Section 7.2. These virtual creatures are represented



(a) Rendered virtual ecosystem with shaded blocks. All available block types are shown and annotated. Vegetation is visible as well.



(b) Rendered virtual ecosystem with shaded blocks. The wireframe is visualized to show the modular, block-based structure.

Figure 6.1: A rendered view of an example virtual ecosystem. This is a rather small and simple environment, given that it only consists of 102×102 blocks and has a flat surface. (a) shows this virtual ecosystem with the different types of blocks annotated. (b) emphasizes the modular, block-based structure of the environment by showing a wireframe version of (a).

by an indirect genetic encoding, a *Compositional Pattern Producing Network (CPPN)*, and simultaneously evolve both their morphology and neural network based controller using the *One CPPN to Rule them All (OCRA)* approach proposed in this thesis (cfr. Chapter 3). The second and somewhat inferior type of organism inhabiting the virtual ecosystem serves as the main food source of the virtual creatures and is portrayed as vegetation. The vegetation is also genetically encoded and subject to evolution, although at a much higher level. Vegetation is further described in Section 7.1.

6.1.1 Simulation Flow

Our general simulation flow follows the same scheme of PolyWorld [25], which was discussed in Section 2.2.2. Simulations here are considered successful if and only if the population of virtual creatures develops a *Successful Behavior Strategy* (SBS); these are populations which are capable of sustaining their numbers through their own mating behaviors (cfr. Section 7.2.3) and thus no longer require any additional artificial creations to maintain a minimum amount of creatures. This definition of success allows to break the simulation down into three main phases: (1) initialization, (2) pre-SBS and (3) post-SBS.

Initialization The initialization phase is responsible for building the virtual ecosystem and inserting an initial seed population of both the virtual creatures and the vegetation. Therefore it uses a user-defined or randomly generated *world-file* that describes: (1) locations of all blocks types (earth, water, wall and random blocks) and (2) locations of the seed vegetation. We choose to use such a world-file to make it easy for users to define or randomly generate their own virtual ecosystems, given that such a world-file comes down to a simple two- or three-dimensional matrix representing the placement of all (non-virtual creature related) blocks in the world.

After the world has been created, the vegetation is added first such that it can already begin to grow and spread (cfr. Section 7.1). While all vegetation in the initial seed population has the same genetics, the initial seed population of virtual creatures do not. Given that we employ the *OCRA* methodology to evolve them, their genomes are *Compositional Pattern Producing Networks* that initially only contain random input to output neuron connections (and no hidden neurons).

Pre-SBS The second phase then tries to stimulate the population of virtual creatures towards obtaining a SBS by running in an "on-line Genetic Algorithm (GA)" mode. Just as in PolyWorld, a minimum number of virtual creatures is guaranteed to populate the virtual ecosystem. If the number of creatures drops below this minimum, we insert a new "batch" of artificially created virtual creatures. Which creatures are inserted depends on the underlying evolutionary framework used. For instance if the MAP-Elites framework [41] is employed (cfr. Section 2.1.3), this new batch would consist of mutated and crossed over versions of the *fittest* creatures encountered up to that point. In this domain, such a fitness function could be defined based on the ability of creatures to survive and reproduce, thereby stimulating the creatures towards reaching a SBS. It is important to note here that this fitness function in essence would only be used to "kick-off" evolution and to steer it towards evolving creatures that sustain their own numbers through self-reproduction. This means that at some point, more specifically when the SBS has been reached, this explicit fitness function will no longer be used as there are no more artificial creations. The virtual creatures are from then on "free to invent their own fitness function" [22]. However, one could argue that this limits the open-endedness of evolution as well, given that the process would still be seeded by a user-defined notion of performance. This can be addressed by replacing fitness based selection with the novelty based selection [24] described in Section 2.1.2.

Post-SBS Once an SBS has emerged, we move into the third phase. Here, the backbone GA is turned off and the evolutionary process is "set free".

6.1.2 Analysis

To allow the human experimenter to observe the current state of the simulation, a set of graphical displays of time histories of certain quantities of interest are made available. These currently include (1) the natural birth to artificial creation ratio (denotes the progress towards an SBS), (2) the absolute amount of virtual creatures alive, (3) the average and maximum amount of time steps lived by the creatures and (4) the absolute amount of vegetation alive. Additionally, depending on the underlying evolutionary framework used, visualizations concerning the average, minimum and maximum fitness or novelty scores, together with both genotypic and phenotypic creature complexities are made available as well.

While limited, these provide a basic overview of both the world state and the evolutionary progress of the virtual creatures. Nevertheless, many other visualizations can be included in the future to gain additional insights from the simulations.

6.2 Unity

As briefly discussed in Section 5.2, the simulator used throughout this thesis is Unity [113]. Unity was originally developed as a game engine but in the recent years broadened its use cases, including applications for the automotive, transportation and manufacturing industries. Additionally, due to the release of the Unity ML-Agents Toolkit [114], it can be used as an environment to train intelligent agents.

While every simulator has its limitations, especially when working towards open-ended evolution, taking such a popular game development platform decreases risk of deprecation and

guarantees some level of extensibility. Moreover, Juliani et al. [115] argue that modern game engines are uniquely suited to act as general platforms and as a case study examine the Unity engine and Unity ML-Agents Toolkit, thereby demonstrating its advantages such as its strong backbone physics engine.

One additional advantage of game engines within this scenario, is their ability to scale. Given the whole *Massively Multiplayer Online (MMO)* genre within games, scalability is often baked right in the game engines. In our case, this scalability is necessary to allow larger population sizes. In biological literature, the concept of *"minimal viable population"* refers to the lower bound on population size such that a species can survive in the wild [21]. Theoretical reasons for such a minimum viable population size include inbreeding and lack of genetic diversity, next to demographic and environmental stochasticity. While it can be tricky to apply such empirical results from the biological world directly to a virtual world like ours, these factors do serve as a warning that a limited capacity to support large population sizes may be a problem. Unity's inherent scalability allows to mitigate this issue.

CHAPTER 7

VIRTUAL ORGANISMS

The virtual ecosystem presented here is home to two types of virtual organisms: *vegetation* and *virtual creatures*. Given that these are both genetically encoded, they are both subject to evolution, although at different levels of complexity. The link between these two types of organisms is the *"energy"* they contain; vegetation being the primary (although not only) food source for the virtual creatures. The main interaction between vegetation and virtual creatures being that the creatures can eat vegetation. Furthermore, an additional set of interactions originates from the ability of creatures to move vegetation around.

As will be elucidated throughout this chapter, these interactions, combined with the genetic variation induced by the evolution of both of these virtual organism, serve as an additional stimulus for the evolution of both of these organisms.

7.1 Vegetation

The primary food source of the virtual creatures is portrayed as vegetation or plants. Within this virtual ecosystem, plants can be seen as a virtual organisms that are subject to evolution as well; although at a much higher level than the creatures. As shown in Figure 7.1, plants here are visualized as trees. Their tree shape makes them the only, non-block based entity within the virtual ecosystem. However, their (invisible) *collider*[1] is still box-shaped, allowing creatures to manipulate them just like any other block within the environment.

1. A collider denotes the shape of an object for the purposes of physical collisions.

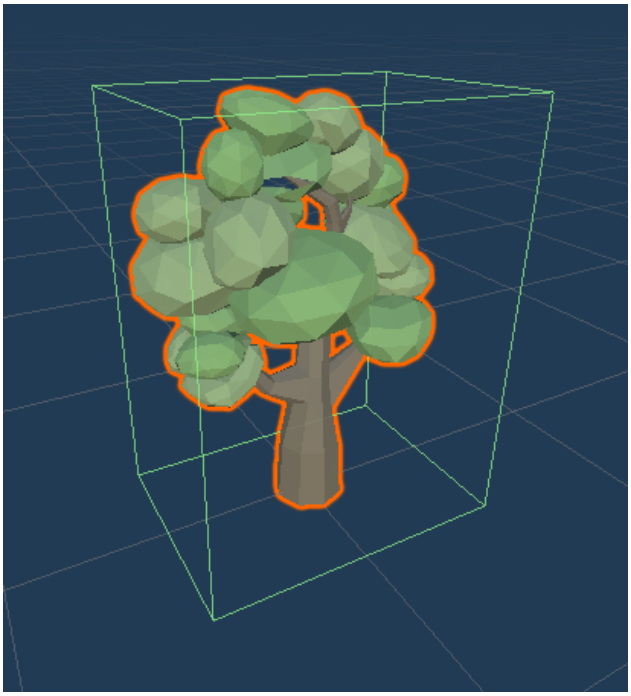


Figure 7.1: Tree model used to portray vegetation in the virtual ecosystem. The model was taken from a publicly available package [120]. The plant's box-shaped collider is shown as well.

7.1.1 Genes

To keep plants, and their evolutionary processes, as simple as possible, they only have one gene: *food type*. This is represented a numerical value in the range of 0.0 to 1.0. As discussed in Section 7.2.2, a plant's food type influences the amount of energy that a creature is able to take from it in one *eat* action, based on its similarity with the *"food type preference"* gene of that creature. This value of the food type gene of plants also defines their color: 0 being depicted as blue, 0.5 as green and 1 as red.

All plants in the initial seed population are instantiated with their food type gene set to 0.5. As described below in Section 7.1.2, a plant can reproduce (asexually) if it has accumulated enough energy. If a plant reproduces, a new plant is instantiated of which the food type gene is a mutated version of the one defined in the original plant. Such mutation is

implemented by stochastically adding or subtracting 0.001 from the original food type value. Consequently, vegetation is subject to a simple type of (purely stochastic) evolution.

The main idea is that this will naturally cause different types of vegetation to occur in different areas of the virtual ecosystem. This in turn establishes an additional incentive for the evolutionary process of the virtual creatures. Given that some types of vegetation will be more advantages to be near to (based on the food type preference of the creature), the creatures are stimulated towards evolving additional complexities such as learning what type of food they "digest" best. Additionally, the eating behavior of the creatures can stimulate the variation within the vegetation, as some food types may become popular and therefore will be targeted by creatures more often. Other types may then be eaten less, allowing them to *spread* more. This in turn stimulates the creatures evolving towards other food type preferences; leading to a never ending cycle of evolutionary stimuli (analogous to what can be observed in our biological world).

However, this requires the creatures to be able to perceive the food type of the vegetation around them. While this could be hard-coded[2] through additional observations given to the creatures, the most straightforward and generic way to do so would be through allowing creatures to have visual observations. Nevertheless, this is not yet supported in the current implementation of the virtual ecosystem (given that this would first require testing the OCRA methodology with creatures using such visual observations), but as described in Section 9.2.2 this establishes one of the most important future works.

2. Relying on hard-coded features is directly opposed to our design goals, as it restricts the open-endedness property of evolution to "what has been programmed in".

7.1.2 Energy

As mentioned before, vegetation forms the primary food source of the virtual creatures. In essence, the plants contain *energy* that the creatures can extract by *eating* them. A plant's energy is thus related to the *food-energy* of creatures, as further described in Section 7.2.2. A plant's energy is represented as a numerical value, ranging from 0.0 to 1.0.

On creation (either as a product of reproduction or as part of the seed population) a plant has an energy value of 0.1. Over time, this energy value increases at a constant rate until the maximum of 1.0 is hit. The actual size of the plant scales with its energy value, thereby visually denoting its energy value[3]. Every time step in which this energy value is above some threshold (e.g. 0.80), it is allowed to reproduce, thereby donating 0.1 energy to its offspring which will be created in a randomly picked (free) location around the original plant. Additionally, the plant loses 0.2 energy which can be seen as an additional *cost* of reproducing; thereby avoiding plants spreading too quickly. This reproduction is done in a stochastic manner, i.e. the decision of whether or not a plant reproduces is purely done by chance. The probability of reproduction increases with the amount of energy available in the plant. Next to the energy transfer to offspring, the energy value of a plant can also decrease when its consumed by creatures. If this causes the energy value to reach zero, the plant is fully eaten and is removed from the environment.

A user-defined limit on the amount of vegetation can be set. This value can range from 0.0 to 1.0 and is defined as the ratio of amount of plants over the total available plant locations (which is currently limited to the earth blocks). In this way, a value of 0.3 indicates that 30% of the earth within the environment may be covered by vegetation. Figure 7.2

3. This forms another reason as to why visual observations are an important next step for the virtual creatures.

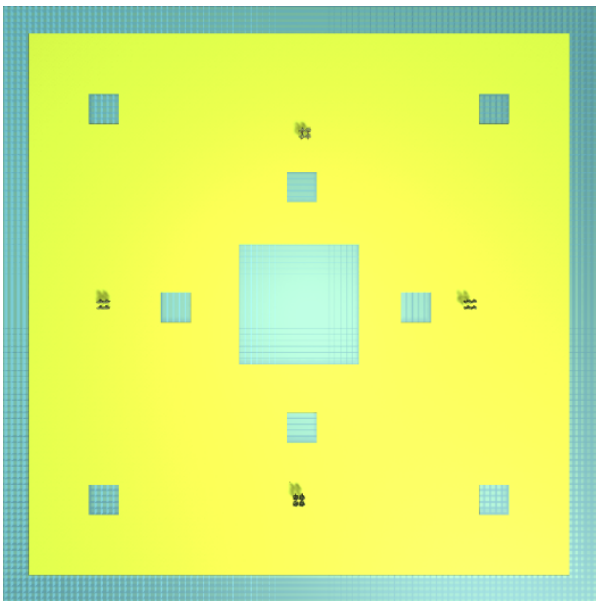
illustrates the spread of vegetation over time.

As mentioned above, creatures can use their morphology to move plants around. During such movement, the plant does not gain any energy and may not reproduce. Only when the plant's location is fixed for some number of time steps, it will again start to grow and spread. In essence, this mechanism allows creatures to bring their food source along to new locations.

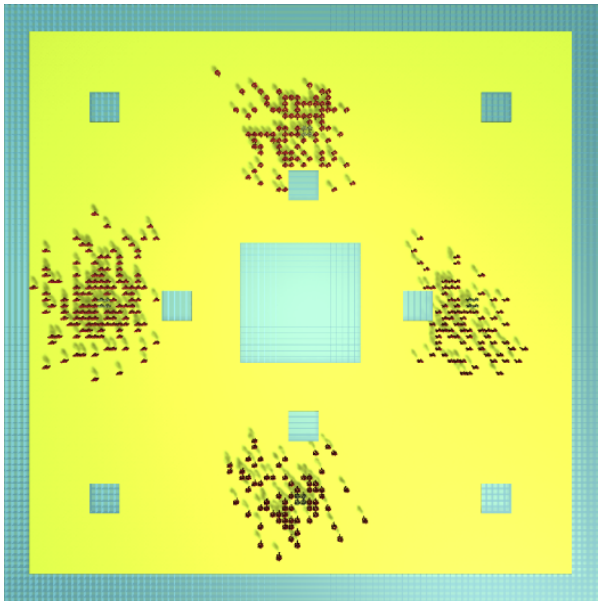
7.2 Creatures

The virtual creatures are the primary inhabitants of the virtual ecosystem and form the main object of study. These creatures simultaneously evolve their morphology and controller through the *One CPPN to Rule them All* (OCRA) approach proposed in Part I. Analogous to the environment itself, these creatures are composed out of a set of predefined building blocks. Their complete phenotype, i.e. the morphological configuration of these building blocks and the neural network based controller, is completely both encoded by their genome: the *Compositional Pattern Producing Network* (CPPN).

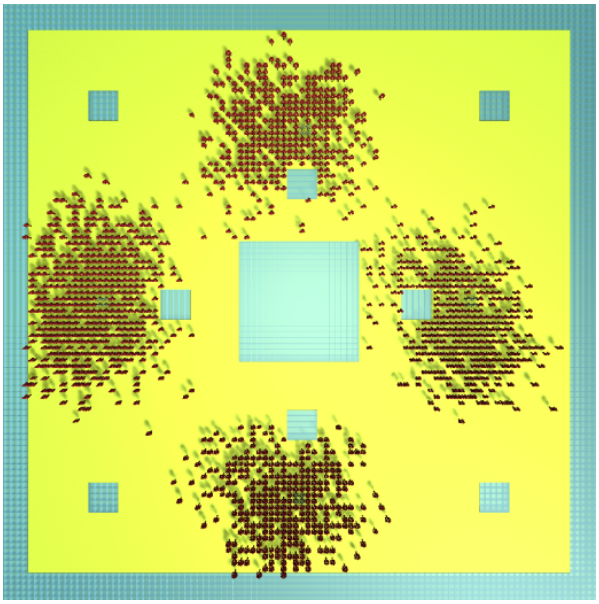
One of the attractive advantages of completely evolving creatures, is that it allows to forgo the necessity of a predefined and generic set of actions that the creatures can take, given that such actions can now be discovered by the creatures themselves. To illustrate, imagine an environment containing a climbable mountain. As commonly seen in many works to date, the fixed morphology of the creatures within such an environment would require making some "hard-coded" action available to allow them to climb the mountain. However, if we let go of the restrictiveness of a fixed morphology, the creatures can discover such an action themselves by evolving towards a morphology capable of climbing, together with a controller capable of steering that morphology towards climbing the mountain. This phenotypic free-



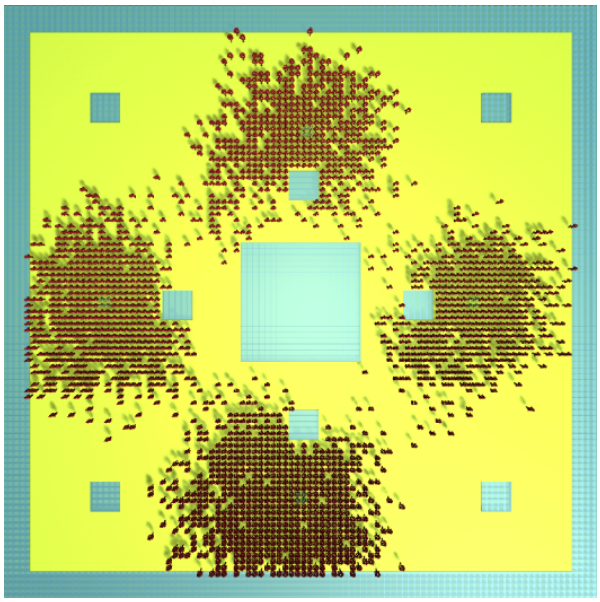
(a) Initial seed population consisting out of 16 plants.



(b) Vegetation coverage ratio: 0.05



(c) Vegetation coverage ratio: 0.2



(d) Vegetation coverage ratio: 0.3

Figure 7.2: Visualization of spreading vegetation over time. The initial seed population (a) is colored green. All offspring is colored red for visibility. Every figure denotes the ratio of amount of plants over the total available plant locations.

dom, combined with a modular virtual ecosystem composed out of mutable building blocks, establishes the major step taken here towards a higher level of open-ended evolution.

Nevertheless, some actions currently employed in the virtual ecosystem presented here are still predefined, including *eating*, *drinking*, *mating* and *attacking*. These actions are made available to the creatures by predefined output neurons in their neural network based controllers. While we do believe that in essence, these could all be incorporated through certain morphological dynamics[4], this would make it much harder for the creatures to survive (given that they would first have to discover all of these dynamics). Consequently, it makes sense to (initially) forgo such additional complexity.

Given that the *eating* and *drinking* actions are related to a creature's energy consumption, these will be described in Section 7.2.2. The *mating* and *attacking* actions are related to interactions between creatures and are therefore described in Section 7.2.3.

7.2.1 Genes

As described above, the creature's complete morphology and neural network controller are encoded by the CPPN genome. The resulting phenotypic configuration of a creature directly affects some additional genetic properties as well. Although these will be further detailed in the subsections below, a brief overview is given here:

- **Attack Power**: The amount of damage done in a single attack. Increases with increasing morphological complexity[5].

4. *Attacking* (other creatures) could for example be incorporated by allowing creatures to break other creatures' morphologies by smashing into their opponents.

5. Morphological complexity refers to the amount of building blocks that the creature's morphology is composed of.

- **Attack Range**: The maximum range of an attack. Increases with increasing morphological complexity.

- **Mating Range**: The maximum range of a mating attempt. Increases with increasing morphological complexity.

- **Maximum energy storage**: The maximum amount of energy a creature can store. Increases with increasing morphological complexity.

- **Energy expenditure**: The amount of energy a creature spends when doing an action (both predefined and morphology dependent actions). Increases with increasing neurological[6] and morphological complexities.

- **Maximum lifespan**: The maximum amount of time steps a creature can live. Increases with increasing morphological complexity.

Additionally, we can introduce a set of higher level genes to allow supplementary creature traits to arise. These are all represented by a numerical value in the range of 0.0 to 1.0. Creatures belonging to the initial seed population have random values for these genes. Mutations on these high-level genes stochastically add or subtract some small constant (e.g. 0.001). When crossing over two creatures, the average value of the high-level genes of the two creatures is taken. The following are (currently) included:

- **Food type preference**: All food sources (both vegetation and other creatures) have a related *food type* gene. The similarity between the value of that food type gene and the value of this food type preference gene influences the amount of energy that a creature is able to take from it in one *eat* action. This can be seen as an abstraction of the digestive ability of creatures.

6. Neurological complexity refers to the amount of hidden neurons and connections within the creature's controller.

- **Food type**: Defines the food type of a creature, which in turn is related to the *food type preference* gene of other creatures. As described in Section 7.2.3, this allows for *food chains* to arise.

- **Morphological resolution**: Defines the resolution at which the creature's morphology is built. This relates to the resolution at which the *Agent Space* is divided into *subblocks* (cfr. Section 3.1). Consequently, this defines the maximum amount and size of building blocks used in the creature.

- **Neurological resolution**: Defines the resolution at which the creature's neural network controller is built. This relates to the resolution at which the *Brain Substrate* is divided into potential hidden neuron locations (cfr. Section 3.2). Consequently, this defines the maximum neurological complexity of the creature.

7.2.2 Energy

Within this virtual ecosystem, creatures have three types of energy: (1) health-energy, (2) food-energy and (3) drink-energy. The maximum amount of energy that a creature can store is equal for all of these types. This amount scales with the morphological complexity of the creature, i.e. the amount of building blocks that its morphology is composed of. The state of all three of these energy types are provided as additional inputs to the controller of the creature, allowing it to change its behavior depending on its energy state. The energy state of each creature is also visualized for the human observer, using colored bars floating above the creature as shown in Figure 7.3.

In order to survive, a virtual creature must keep its health-energy from reaching zero, which can happen due to "starvation", "dehydration" or being attacked by other creatures. Attacks will be further discussed in Section 7.2.3. Whereas starvation occurs when a creature's

food-energy has reached zero, dehydration occurs when a creature's drink-energy has reached zero. Both of these independently reduce the health-energy at a constant rate, such that when both occur simultaneously, these rates are coalesced. Health-energy is regenerated when food- and drink-energy levels are both above zero; thereby draining these two types of energy. A creature dies when it has reached its maximum lifespan (cfr. Section 7.2.1) or if its health-energy reaches zero. This however allows the possibility of dying while still maintaining relatively high food- and drink-energy values (for instance if health-energy was driven to zero due to an attack). A creature's remaining food and drink-energy remains stored in its carcass, such that other creatures (e.g. the attacker) can consume it. As further detailed in Section 7.2.3, this mechanism makes predator-prey interactions quite natural.

Whereas food-energy can be replenished by *eating* vegetation, drink-energy is replenished by drinking from *water* blocks. Both types of energy can be gained simultaneously by *eating* a carcass. As mentioned above, the *eat* and *drink* actions are predefined and are made available to the creatures by predefined output neurons in their neural network based controllers. In order to successfully eat or drink, they must be near the target when activating the corresponding output neuron. While the amount of drink-energy gained by a single *drink* action is constant, the amount of food-energy gained by a single *eat* action depends on the similarity between the creature's *food type preference* gene and the target's *food type gene*. The higher the similarity, the more energy is consumed at once.

Both food- and drink-energy are depleted at a constant "metabolism" rate, as creatures could otherwise live out their full lifespan without taking actions to survive. However, food-energy is additionally depleted by taking actions. The amount of energy expended by *reproduction*, *attacking* and morphological actions, i.e. joint movements, scales with the complexity of both the morphology (amount of building blocks) and the controller (amount

of neurons and connections). While such actions can consequently cause starvation to occur more quickly, if creatures take appropriate actions (e.g. moving towards food and drinks) they can increase their lifespan by responding to the pressure of the constant metabolism rate.

Related to this energy consumption is that during sexual reproduction, both parent creatures transfer some fraction of their food- and drink-energy to their offspring. This can be seen as the "cost" of reproducing and the amount of energy transferred sets the initial energy levels of the offspring. Such a mechanism helps to avoid scenario's in which creatures reproduce with the sole intention of eating their offspring afterwards. Reproduction will be further discussed in Section 7.2.3

7.2.3 Interactions

As pointed out by many of the works within this domain [22, 25, 26, 61], interaction establishes a fundamental drive towards an open-ended evolutionary process yielding increasingly diverse and complex virtual creatures. In line with research objective two presented in Section 1.1, one of the goals of this virtual ecosystem is to inherently incorporate such interactions. These are described below and are categorized in (1) interactions between creatures and (2) interactions between the creature and its environment.

Interactions between creatures

The phenotypic freedom induced by the simultaneous evolution of both creature morphology and controller, by itself, introduces a theoretically unlimited range of interactions between creatures. As described above, it allows to forgo the necessity of a predefined and generic set of actions that creatures can take, given that such actions can now be discovered by the creatures themselves. Consequently, the interaction between creatures can be discovered as well and are therefore no longer dependent on the creativity of the designer. To illustrate,

such an interaction could be *parasitism*. Given that creatures have a gene that denotes their morphological resolution (which directly affects their size) it is possible that next to "ordinary" sized creatures, a variety of smaller creatures are evolved. These creatures could then potentially climb on top of bigger creatures, thereby feeding from the same food sources as their host (given that the host carries them towards the food it seeks for its own survival). This in turn could lead to the bigger creatures evolving some sort of resistance to such parasites, by physically shoving them of their morphology. Although we do realise that the chance of this happening is substantially small, the important thing is that its possible; especially when designing an environment with open-endedness of evolution in mind.

Nevertheless, some basic actions were predefined to (initially) lower the complexity threshold of creatures surviving in the virtual ecosystem. These include *eating*, *drinking*, *attacking* and *mating*. Whereas the *eating* and *drinking* actions were described in Section 7.2.2, the *attacking* and *mating* actions are described here, given that these respectively introduce two important interaction related concepts: *predation* and *sexual reproduction*.

Predation In essence, *predation* is the interaction between two organisms in which one organism kills and consumes the other. Drawing inspiration from PolyWorld [25], predation is naturally incorporated in this virtual ecosystem by two underlying features: (1) allowing creatures to attack each other and (2) the separation of health-, food- and drink-energy. Given that attacks only affect the health-energy of a creature, it is possible for a creature to be killed by having its health-energy driven to zero by attacks, while still maintaining a relatively high food- and drink-energy value for the attacker.

While the attack action could have been morphologically defined, for instance by letting creatures break each others morphologies by smashing into them with their own morphology, this was simplified by introducing an *"attack-laser"*. This is illustrated in Figure 7.3.

This attack-laser comes down to a directed laser beam, which the creatures can "shoot" out of one of the faces of their *brain block*[7]. The laser beam is activated if the activation level of the corresponding neuron (one for each direction) in the controller exceeds some threshold (e.g. 0.5 if it the output ranges from -1 to 1). If the laser beam hits a creature, that creature's health-energy is reduced based on the attackers morphological size (cfr. Section 7.2.1). The range of the laser beam of a creature also depends on its morphological size. In order to avoid larger creatures having a too big advantage, the food-energy cost of attacking scales with this morphological size as well. Such an interplay between opposing advantages is important in order to produce behavioral niches.

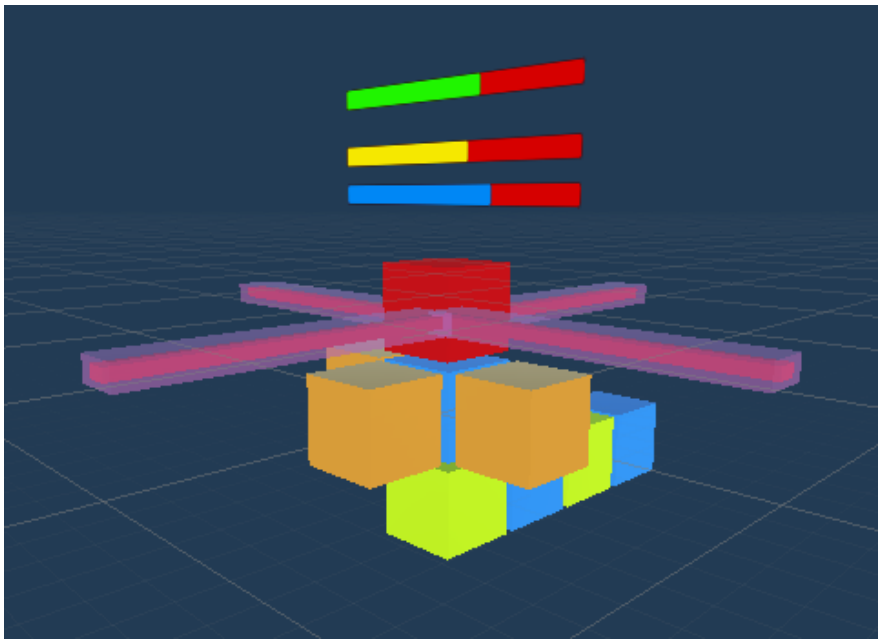


Figure 7.3: Example of a snake-like creature activating all four of its attack-lasers. All of these originate from its brain block (red). Above the creature, three bars indicate its current energy state. From top to bottom respectively: health-energy (green), food-energy (yellow) and drink-energy (blue).

Some regulation concerning predation is naturally incorporated by making the energy storage capacities of creatures dependent on this morphological size as well. Smaller crea-

7. The brain block is a unique building block incorporated in each creature morphology, which contains the spatially organized controller of the creature (cfr. Section 3.2).

tures can be easily killed by larger creatures as larger creatures are able to do more powerful attacks. However, killing such smaller creatures may not be profitable for the larger creatures, given that they may store less food- (and drink-energy) than what the attacker pays for its attack.

In a final remark concerning predation, we would like to point out that this mechanism, combined with the higher-level *food type* and *food type preference* genes of creatures, in essence could bring rise to *food chains*. Some area within the environment could for example only contain vegetation of a certain food type. This area could then attract creatures with a matching food type preference. Given that those creatures also represent some distinct food type, they could in turn attract other creatures that prey on them; even though the vegetation does not suit their food type preferences. Consequently, a three-component food chain has emerged.

However, this requires creatures to be able to perceive the food type of other creatures; again indicating the need for more generic creature observations, such as the visual observations discussed in Section 9.2.2.

Sexual Reproduction Self-reproduction, i.e. requiring organisms to build their own offspring rather than employing an extrinsic mechanism to decide which organisms can reproduce, establishes an important requirement towards open-ended evolution [21]. The importance of self-reproduction is fundamentally incorporated within the virtual ecosystem presented here, given that we consider a simulation successful if and only if the population of virtual creatures develops a *Successful Behavior Strategy* (cfr. Section 6.1.1). Such a population is capable of sustaining their numbers through their own reproductive behaviors and thus no longer requires any artificial creations to maintain a minimum amount of creatures. Reproduction can be either sexual or asexual. Whereas sexual reproduction involves two

organisms, asexual reproduction involves only one. Within the presented virtual ecosystem, (currently) only sexual reproduction is enabled.

Therefore, a laser based approach similar to the attack procedure described above is employed. This *"mate-laser"* is activated if the activation level corresponding to the *mate* action of the creature exceeds some threshold. To initiate reproduction, two creatures must hit each other with their mate-lasers simultaneously. Consequently, both creatures have to "express" the desire to mate.

If such an event occurs, the two creatures are first compared and the strongest (in terms of remaining energy levels) is designated as the *carrier* of the offspring. Afterwards, the gestation period is started and both parent creatures transfer a fraction of their food- and drink-energy to the future offspring. This can be seen as the cost of reproduction and the amount of energy transferred sets the initial food- and drink-energy levels of the offspring[8]. The offspring is a single creature of which the genome (the combination of the CPPN and the high-level genes described in Section 7.2.1) is a crossed over[9] and mutated version of the genomes of both parents. The offspring's phenotype is built during this gestation period using the OCRA approach discussed in Chapter 3. While the "child" creature is being built, the controllers of both parents receive the direction to each other as additional observations (this for instance could allow the non-carrier to stay near the carrier for protection). Furthermore, the carrier's controller is informed of its role with an additional *pregnant*-flag observation.

If the carrier dies during the gestation period, the reproduction has failed and no offspring

8. Health-energy is always initialized as the maximum amount.

9. This crossover thereby incorporates the vertical mixture of genetic material, which was indicated as one of the goals of research objective two (cfr. Section 1.1).

will be created. If the carrier survives, the offspring is inserted into the environment near the carrier when its construction is finished. Per reproduction event, only one creature is created. The controller of the offspring receives the direction to both of its parents as additional observations. The controllers of both the parents from then on also receive observations describing the direction to their offspring, however the direction to each other is no longer provided. All of these additional observations rely on the underlying neuroevolution algorithm's capacity to evolve controllers capable of processing such information, thereby doing appropriate feature selection as witnessed in the sensor related OCRA experiments (cfr. Section 5.3.3).

Reproduction could be further regulated by introducing a *miscegenation function* (so dubbed by R. Dawkins) as employed in PolyWorld [25]. This would allow to probabilistically influence the likelihood of genetically dissimilar organisms producing viable offspring; the greater the dissimilarity, the lower he probability of successful reproduction. However, given that this makes it harder for a Successful Behavior Strategy to emerge, this is currently not employed.

Interactions between the creature and its environment

Analogous to the case of interactions between creatures, the phenotypic freedom induced by the simultaneous evolution of both creature morphology and controller, introduces a vast range of possible interactions between the creature and its environment. This range is further expanded by the modular, i.e. block based, design of the environment. This design for instance allows creatures to change the environment by moving the blocks around.

One type of environment block, the *randomly defined block*, is particularly designated for

this purpose. The randomness relates to their geometry[10], i.e. their size and shape. Upon initialization of the environment, a user-defined amount of such randomly generated blocks is inserted. These can then be used by the creatures for whatever they see fit, such as potentially constructing "barriers" capable of holding off predators. With only a slight increase of computational complexity introduced by simulating these additional objects, this opens up a whole range of new possible activities for the creature. Consequently, it can be seen as an additional step towards open-endedness. To the best of our knowledge, this is the first virtual ecosystem that incorporates this concept.

10. While it would be interesting to include randomness in the density of such blocks as well, thereby introducing "soft" blocks next to the rigid blocks, this is currently not properly supported by the simulator.

CHAPTER 8

EXPERIMENTS

This chapter discusses the experiments carried out within the proposed virtual ecosystem. However, these experiments all took place in simplified versions of the virtual ecosystem. The main reason for this simplification being the novelty of the approach used to evolve the virtual creatures. This is the *One CPPN to Rule them All* (OCRA) approach proposed in Part I of this thesis. Given its novelty, we cannot expect it to immediately take on the complexity of the complete virtual ecosystem and lead to successful results. Additionally, we believe in an *iteration-based* modus operandi, in which each iteration of experiments adds only a minor amount of additional complexity compared to the previous one.

Nevertheless, the complete virtual ecosystem that was presented in the previous chapters is fully implemented and is ready for further experimentation[1].

8.1 Validating and Debugging through Reinforcement Learning

During the development of the virtual ecosystem, we employed a constant evaluation loop. The evaluators in this case were Reinforcement Learning[2] agents. Although they were given a fixed morphology and a completely predefined set of actions (cfr. Section 8.2), they were given the same goal as the virtual creatures: reaching a Successful Behavior Strategy.

In practice, whenever a new group of features was incorporated within the virtual ecosystem, such as the growth and spread of vegetation, an experiment was instantiated in which RL driven agents were given the goal to survive and reproduce well enough such that the amount

1. The virtual ecosystem is available at: `https://github.ugent.be/dmarzoug/thesis`

2. Reinforcement Learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward [121].

of agents always stays above some minimum threshold. Based on their inherent exploration capacity, these agents then (more often than not) shed light on flaws and bugs within the environment during their optimization process.

Given that Reinforcement Learning is not the object of study here, we will not go further into detail concerning its exact usage. However, we do want to note that all RL agents shared the same neural network brain, which was trained using the Proximal Policy Optimization (PPO) algorithm [122].

8.2 Validating the Neuroevolution Algorithm

The first series of experiments were used as an additional validation of the neuroevolution algorithm employed in the *One CPPN to Rule them All* approach. This neuroevolution algorithm is the *Adaptive Evolvable-Substrate HyperNEAT* (AESHN) algorithm, discussed in Section 2.5.4. Furthermore, the modifications to this algorithm proposed in Section 3.2 were included.

The Environment

As mentioned above, the experiments presented here employed simplified versions of the presented virtual ecosystem. The simplifications in this case included the exclusion of the *food type* gene dynamic within both the creatures and the vegetation. Furthermore, a flat landscape was used without any *randomly defined* blocks. The goal was no longer to reach a Successful Behavior Strategy, instead the creature was only given the goal to survive as long as possible through eating vegetation and drinking from water blocks. In other words, no interaction with other creatures such as sexual reproduction was required. Consequently, this allowed us to evaluate each candidate solution independently from the others by placing it alone in the environment; thereby avoiding the additional complexity of creatures influencing

each other. The environment is shown in Figure 8.1.

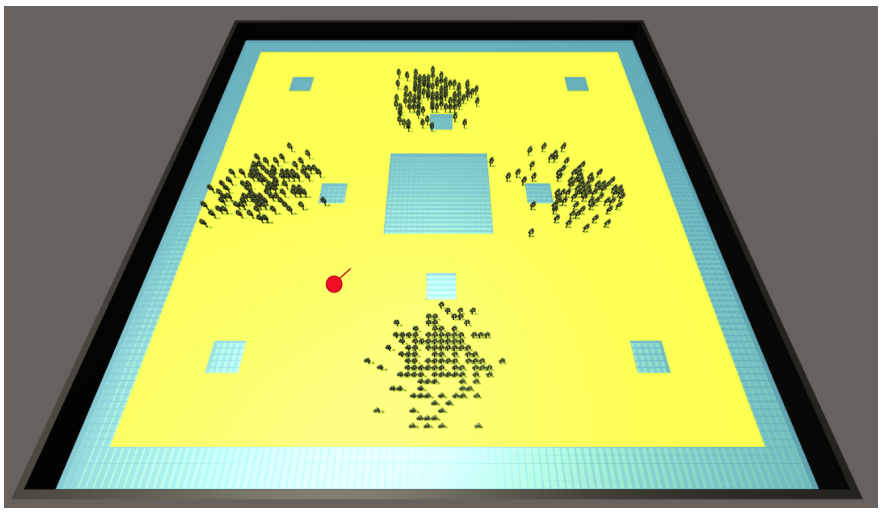


Figure 8.1: The simplified virtual ecosystem used for additional validation of the AESHN neuroevolution algorithm. The creature's starting location and orientation are shown in red.

Experimental Setup

Given that only the neuroevolution algorithm algorithm was tested here, the creatures' morphologies were fixed. This morphology is the same as the one used by the RL agents described above and comes down to a movable cube, as shown inFigure 8.2. Consequently, all creature actions were predefined as well. These were organized in groups, such that at most one action per group could be simultaneously executed. The actions can be summarized as follows:

- **Forward motion**: move forward or backward.
- **Side motion**: move left or right.
- **Rotation**: turn left or right.
- **Act**: jump[3], eat or drink.

3. The *jump* action is necessary to allow the creatures to jump out of the pools of water blocks.

To perceive their environment, the creatures were given two rows of *ray* sensors. This is the same type of sensor as used in Section 5.3. In essence, this sensor casts an array of rays into the environment, that each return the type of object they collide with. Additionally, the distance to that object is returned. These rays are visualized in Figure 8.2. As can be seen, one row of rays is cast straight ahead. These allow the creature to see what type of environment blocks are in front of it, next to possible vegetation. Furthermore, one row of rays is cast in a downward angle. These allow the creature to see if there is either an earth block or a water block right in front of it, which is beneficial to appropriately do the *drink* action.

Next to these ray sensor observations, the evolved controller of the creatures were provided with three additional observations that indicate the current *health*-, *food*- and *drink*-energy state of the creature. Lastly, three additional observations describing the creature's motion state were provided. These indicate the current velocity of the creature in the x, y and z direction.

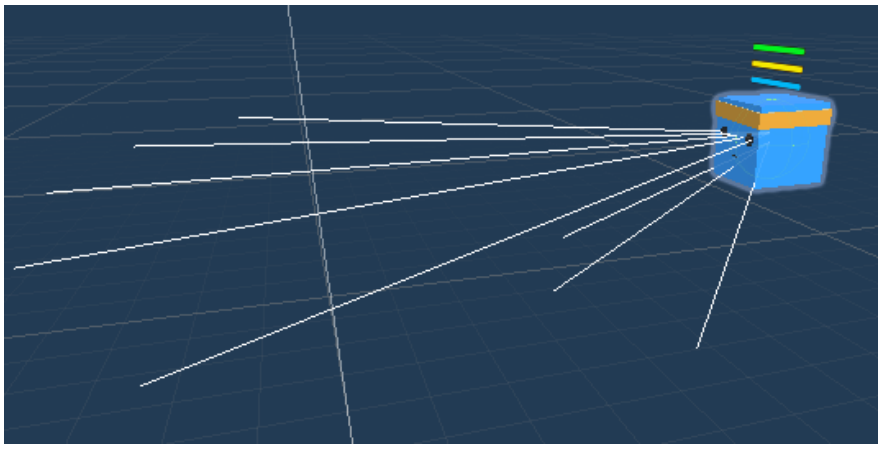


Figure 8.2: Illustration of the fixed creature morphology and ray sensor configuration. The cube-shaped model of the creature morphology was taken from Unity's ML-Agents ToolKit [114]. Above the creature, three bars indicate its current energy status. One row of 5 rays is cast straight ahead, allowing the creature to see the type of environment blocks or vegetation in front of it. The other row of 3 rays is cast in a downward angle, allowing the creature to see if either an earth or water block is in front of it.

Just like in the neuroevolution experiments described in Section 5.1, the NEAT algorithm (cfr. Section 2.5.2) was used to evolve the CPPN genomes. Such a CPPN genome is then used by the AESHN algorithm to build candidate solutions (i.e. neural network controllers). Novelty Search [24] was used as the evolutionary search algorithm. As discussed in Section 2.1.2, to calculate the novelty score of a candidate solution, we require a *behavior descriptor* that describes the candidate solution's functional behavior. In this experiment, the functional behavior of a creature was represented by a *time series* of 20 sampled creature energy levels. The key insight behind this being that this encapsulates both the creature's energy accumulation (eating and drinking) and depletion (performing actions expends energy) over time. Therefore it presents an abstract view of the creature's behavior during simulation.

Population size was set to 300 with an elitism of 2. CPPN mutation probabilities were set to 0.03 and 0.005 for connection additions and removals, 0.02 and 0.005 for neuron addition and removals and 0.94 for connection weight updates. By keeping the structural mutations rather low compared to the connection weight updates, we allow the evolved CPPN genomes some time to adapt before structurally mutating them.

Given that no morphological evolution is incorporated here, the CPPN genomes use only controller specific input and output nodes. This CPPN configuration is the same as described in Section 5.1 and was shown in Figure 5.2. The substrate, i.e. the space in which we let AESHN evolve a neural network controller, is chosen to be a two-dimensional rectangle. This rectangle is centered at origin and covers the range $[-1, 1]$ in both the horizontal x and vertical y dimensions. The spatial organization of the predefined input and output neurons within this substrate is shown in Figure 8.3. The variance thresholds (VT) used by the AESHN algorithm were set to 0.5 for the division phase, 0.2 for the pruning phase

and 0.5 for the band-pruning phase. The minimum substrate resolution was set to 3, while the maximum was set to 4. This corresponds to the minimum and maximum depth of the quadtree representation of the hypercube (cfr. Section 2.5.4), leading to a maximal neural complexity of 256 hidden neurons. The activation function within the evolved neural network controllers was set to the hyperbolic tangent function.

On a final note, the experiment was run on an *Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz*, using multiprocessing to take advantage of the 8 cores available. Table 8.1 summarizes the experimental setup used for the results discussed below.

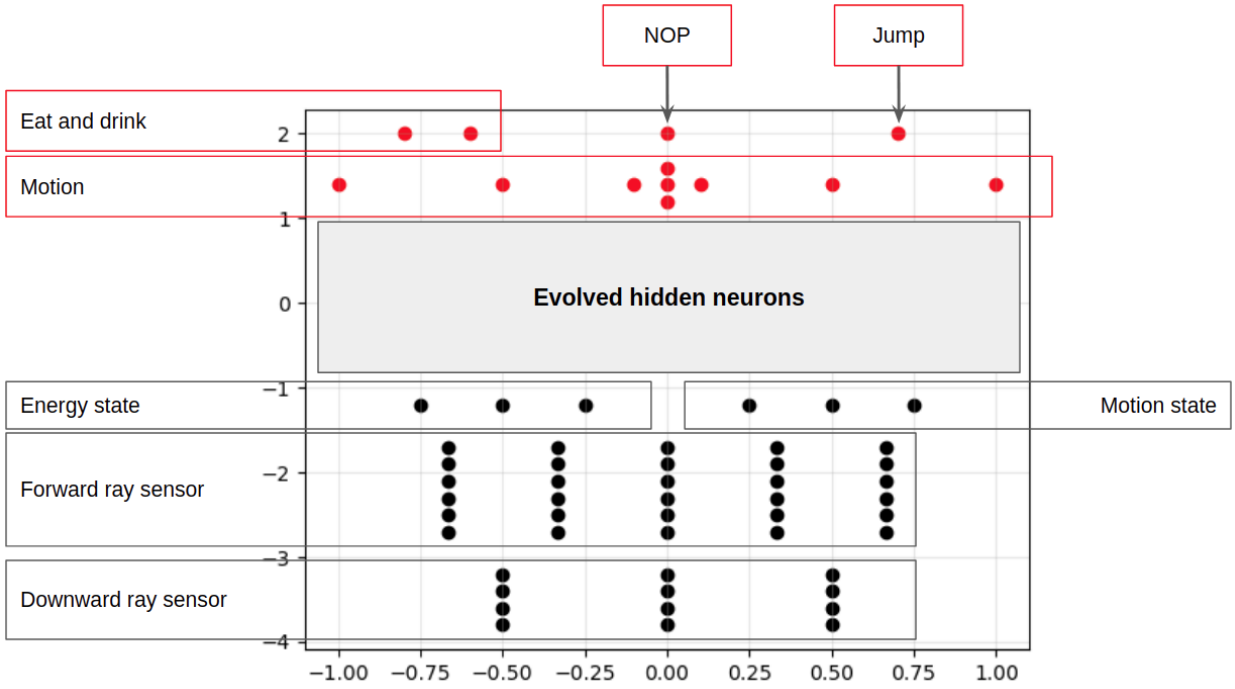


Figure 8.3: The spatial organization of the predefined input (black) and output (red) neurons within the two-dimensional brain substrate. The input neurons corresponding to the different rays are ordered from left to right in the same way that they are cast out in the environment. Analogously, the output neurons corresponding to motion are organized similarly to the direction in which they cause the creature to move. As discussed in Section 2.5.3, this spatial organization allows to inject the geometry of the problem into the structure of the ANN based controller. The space in between the input and output neurons is reserved for the hidden neurons and connections evolved by the AESHN algorithm.

CPPN mutation probabilities	Neuron addition / removal	0.02 / 0.005
	Connection addition / removal	0.03 / 0.005
	Connection weight update	0.94
AESHN	Division phase VT	0.5
	Pruning phase VT	0.2
	Band-pruning phase VT	0.5
	Min / Max substrate resolution	3 / 4
Evolutionary Search	Algorithm	Novelty Search
	Population size	300
	Behavior descriptor	Energy time series
	Elitism	2
Computational Setup	CPU	Intel Core i7-4790
	Total number of cores	8

Table 8.1: Experimental setup of the "Validating the Neuroevolution Algorithm" experiment.

Results and Discussion

Performance of a candidate solution, in this case an evolved ANN controller, is measured as the number of time steps that the controller is able to keep the creature alive. It can do so by steering the creature towards vegetation and water blocks, such that it can prevent starvation and dehydration. Figure 8.4 shows the highest performance encountered in every generation. As can be seen, the highest performing controller was evolved at generation 162 (equivalent to 48600 evaluations given the population size of 300) and was able to keep the creature alive during 1500 time steps. This controller is shown in Figure 8.5 and contains 132 standard hidden neurons, 32 recurrent hidden neurons and a total of 5080 connections. Vi-

sual analysis of the controller's behavior[4] however indicated that the controller only steered the creature towards vegetation, thereby only replenishing its food-energy and completely ignoring the drink-energy.

Nevertheless, given that the goal of this experiment was mainly to validate the modified AESHN neuroevolution algorithm, these results were considered sufficient.

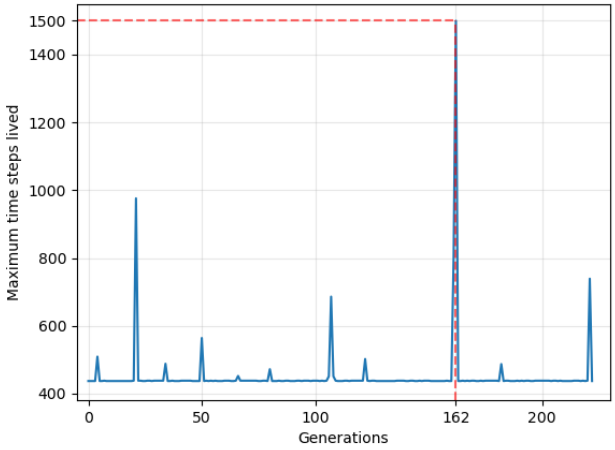


Figure 8.4: The maximum number of time steps lived (y) by a creature in the population of every generation (x). After 4 generations (1200 evaluations), the first controller was evolved that prolonged the minimum lifespan of 437 time steps to 509 time steps by steering the creature towards vegetation and eating it. The best performing controller was evolved after 162 generations (48600 evaluations) and further extended this lifespan to 1500 time steps.

4. A video demonstrating this behavior is available at: `https://www.shorturl.at/hFHT0`

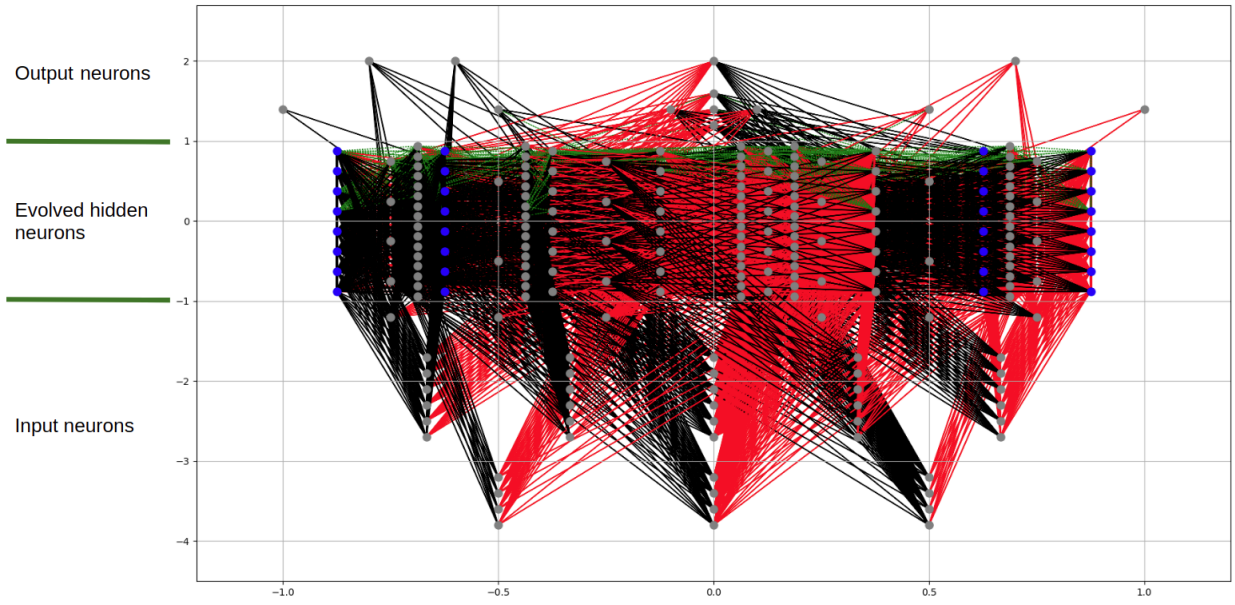


Figure 8.5: The best performing controller that was evolved after 162 generations. This controller contains 132 standard hidden neurons (gray), 32 recurrent hidden neurons (blue) and 5080 connections. Connections are colored black if positive, red if negative and (dotted) green if neuromodulatory. Of those 5080 connections, 444 were neuromodulatory connections. Neuromodulatory connections are used to allow the controller to adapt its weight during lifetime (Section 2.5.4).

8.3 Incorporating OCRA

The final series of experiments done within this thesis were concerned with testing the *One CPPN to Rule them All* (OCRA) approach to evolve complete virtual creatures within the virtual ecosystem. Compared to the previous experiment discussed in Section 8.2, these experiments thus included some additional complexity. Next to evolving a neural network based controller, the morphological evolution was now incorporated as well.

The Environment

In a way to counteract the increased complexity, the environment used was again a simplified version of the proposed virtual ecosystem. The simplified environment used here is similar to the one used in the previous neuroevolution experiment (cfr. Section 8.2). However, some additional simplifications were made. First of all, the drink-energy component of the creatures was removed. Consequently, survival is facilitated as they now only have to focus on finding and eating food (vegetation). Second, given that the creatures no longer have to drink, the water blocks in the environment were replaced by earth blocks. This sidesteps the additional complexity of falling into water and getting stuck. Furthermore, all vegetation had fixed locations and were fully grown from the start. Lastly, the size of the environment was halved in both length and width to reduce the distance that the evolved morphologies have to traverse. The resulting environment is shown in Figure 8.6.

Experimental Setup

The experimental setup employed here is mainly the same as the one used in the *Crawlers*-experiment discussed in Section 5.3.3. Consequently, instead of letting the creatures evolve their own sensor blocks, the sensor were fixed to the brain block. However, while the original Crawlers-experiment only used one ray per horizontal direction, here we increase it to three

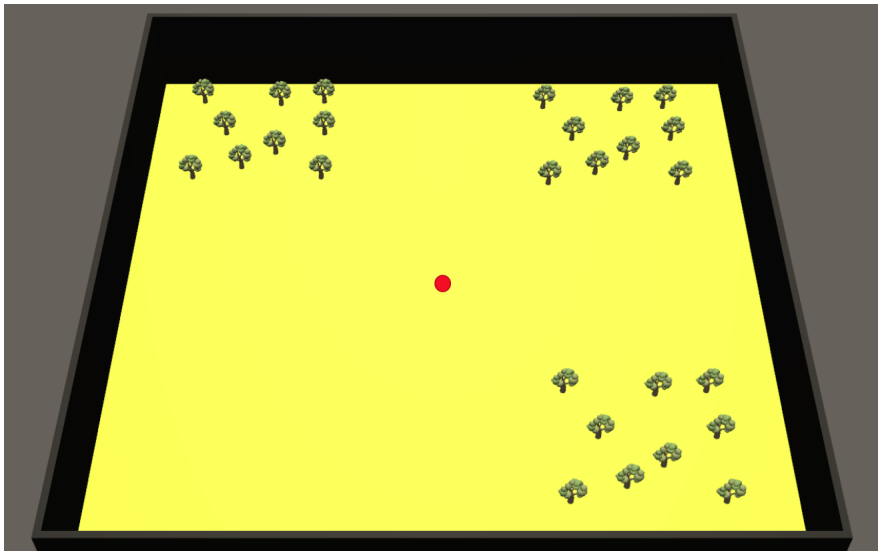


Figure 8.6: The simplified virtual ecosystem used for the OCRA experiments. The creature's starting location is shown in red.

per direction to improve the creature's ability to perceive its environment. Furthermore, an additional *general* output neuron was added that corresponds to the predefined *eat* action and an additional *general* input neuron was added that is given the current *food-energy* state of the creature. The resulting ensemble of predefined input and output neurons is shown in Figure 8.7.

In line with the Crawlers-experiment, CVT-MAP-Elites [45] was used as the underlying framework to guide the evolutionary process (cfr. Section 2.1.3). However, we moved from random selection to *curiosity*-based[5] selection [29]. The idea behind this is to reward individuals that either produce offspring with novel phenotypes or offspring that performs better than the individuals already stored in the archive. In this way, we stimulate the selection of individuals that will most likely produce new cells in the container or improve already occupied cells. The *curiosity score* attempts to do this by modelling the probability of an individual to generate offspring that will be added to the container. Practically, this

5. "Curiosity" within the evolutionary computation domain is not related to curiosity within the domain of Reinforcement Learning [123].

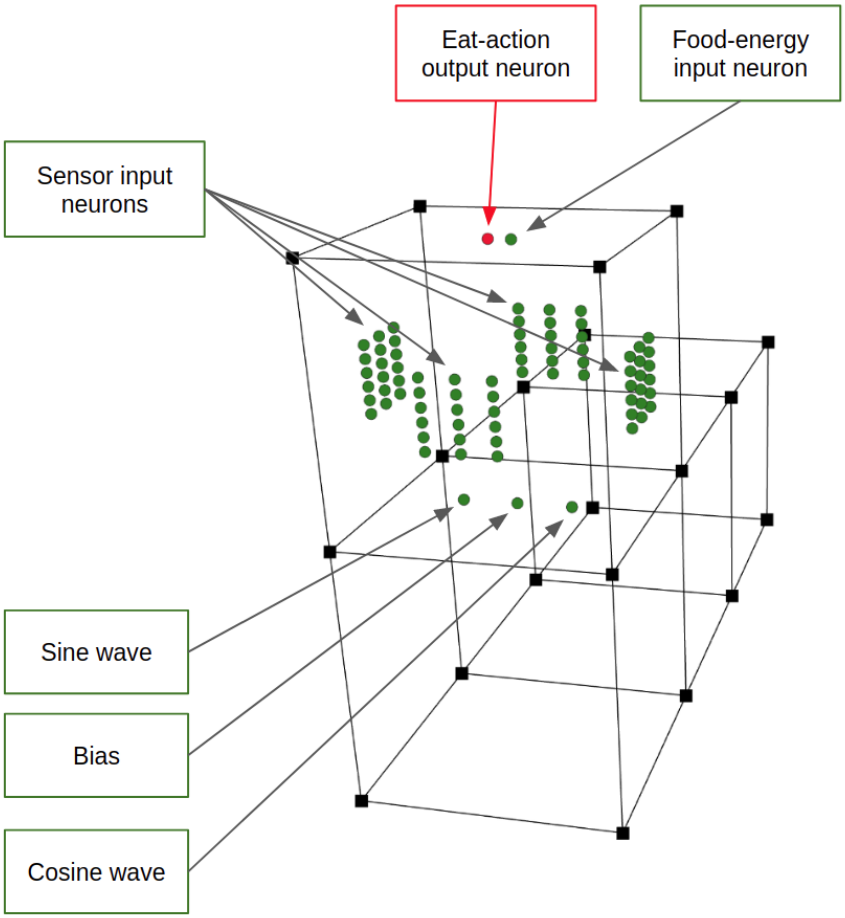


Figure 8.7: Schematic visualization of the placement of the *general input neurons* (green) and the single *general output neuron* (red) within the brain block of a simple crawler. These general input and output neurons are placed in the outer region of the brain block, as the inner region is reserved for the hidden neurons placed by the AESHN algorithm. The general input neurons corresponding to the bias, sine and cosine wave inputs are placed at the bottom. The general input neurons corresponding to the four sensors are placed at their respective face of the brain block, from which they cast three rays outwards. Each of these sensors are coupled with six input neurons per ray that they cast, given that each such ray returns six observations denoting the type of object hit and the distance to that object. The general input neuron corresponding to the *food-energy* observation is placed at the top, with the general output neuron corresponding to the *eat-action* placed next to it.

is implemented by setting the curiosity score of each individual to zero, and then each time that one of its produced offspring gets added to the archive, the individual's curiosity score increases, whereas it decreases each time their offspring fails to enter the archive. Each generation, we can then select individuals from the archive using a weighted probability corresponding to their curiosity scores.

Two runs were carried out, one using the CVT-MAP-Elites approach and one using the *"CMA-ES infused CVT-MAP-Elites"* approach proposed in Chapter 4. This extension aims to mitigate an issue concerning *"the lack of innovation protection"* that arises when the underlying genetic encoding is a neural network such as the CPPN in our case. This lack being the reason why the genotypic complexity in all previous OCRA experiments remained rather low. Additionally, the CMA-ES extension aims to mitigate an inefficiency concerning *"the lack of information reuse"*, which originates from the inherent random- and indirectness of mutations. While these random mutations do carry some benefits such as implementation simplicity, the major drawback is that there is no information reuse between the evaluation results of the different genomes. In other words, nothing is "learned" when a given set of parameters performs well (but not well enough to be placed in the archive) and a different set performs poorly. Additionally, it is possible that the same genome is rediscovered over and over again, causing the algorithm to waste computation by circling back to previously seen (but already forgotten) solutions.

The main idea behind the CMA-ES infused CVT-MAP-Elites approach is to group all CPPN genomes based on their structure (hidden nodes, activation functions and connections). Every time a new structure is discovered, we create and store one CMA-ES instance in an additional archive: the *ES-Archive*. The major difference with the original (CVT-)MAP-Elites algorithm being that instead of selecting genomes, randomly mutating them both

structurally and non-structurally and thereafter evaluating them, we now only randomly mutate them structurally and leave the non-structural optimization for their corresponding CMA-ES instances. Each time some genome structure would be evaluated in the original approach, we now instead continue its stored CMA-ES instance for a user-defined amount of additional iterations.

In this experiment, that amount of additional iterations is set to 1. Given that we use a CVT-MAP-Elites archive size of 100, while selecting 256 parents in every generation, many of the selected parents will be the same. Consequently, the same genome structure will be selected multiple times, each time increasing the number of candidate solutions sampled from its CMA-ES instance and evaluated within that iteration by 4. Each such a candidate solution represents the numerical values of connection weights, node biases and node responses for the given CPPN genome structure. Innovation protection was incorporated by giving new structures, i.e. new CMA-ES instances, 8 samples in their initial iteration. Additionally, every generation the currently 4 least progressed CMA-ES instances are advanced one iteration using 4 samples.

Lastly, the structural mutation probabilities of the CPPN genomes were increased: from 0.03 and 0.005 for connection additions and removals to 0.25 and 0.01 respectively and from 0.02 and 0.005 for neuron addition and removals to 0.20 and 0.01 respectively. This more "agressive" mutation strategy, combined with the information reuse though stored CMA-ES instances, can be seen as an auxiliary innovation protection[6] mechanism; given that this stimulates the (re)discovery of novel structure.

Both runs were conducted on a cluster of eight compute nodes. Each such node containing

6. This somewhat follows the adage - "The best defense is a good offense."

four *Intel(R) Xeon(R) CPU E5-2650 v2 octa-core @ 2.60GHz* CPUs. Multiprocessing was used to take maximal advantage of the 256 cores available. The experimental setup of both of these runs is summarized in Table 8.2.

		Run 1	Run 2
CPPN mutation probabilities	Neuron addition / removal	0.02 / 0.005	0.20 / 0.01
	Connection addition / removal	0.03 / 0.005	0.25 / 0.01
	Connection weight update	0.94	
AESHN	Division phase VT	0.3	
	Pruning phase VT	0.03	
	Band-pruning phase VT	0.3	
	Min / Max substrate resolution	2 / 4	
Evolutionary Search	Algorithm	CVT-MAP-Elites (fitness based replacements, curiosity selection)	CMA-ES infused CVT-MAP-Elites (fitness based replacements, curiosity selection)
	Population size	256	N.A.
	Behavior descriptor	Trajectory + morphology	
	Elites archive size	100	
Computational Setup	CPU	(8 x 4) x Intel Xeon E5-2650	
	Total number of cores	256	

Table 8.2: Experimental setup of the two runs done in the "Incorporating OCRA" experiment.

Results and Discussion

Fitness here is defined as the absolute amount of food-energy replenished by eating vegetation. This stimulates both taking actions (depleting energy, therefore allowing more energy to be regained) and surviving (regaining energy). Additionally, we again incorporate the validity of the creatures (or in this case crawlers) into their fitness score by initially setting it to -2, increasing it to -1 if it has a valid morphology and to 0 if it also has a valid controller.

Figure 8.8 shows the highest fitness of the individuals stored in the CVT-MAP-Elites archive of both runs over time. We chose to compare the runs using the measured wall clock time instead of the number of generations. The reason for this is that the amount of evaluations done in one generation using the CMA-ES infused approach fluctuates over time. Consequently, comparing them on a generational level would be unfair given that the amount of evaluations done after each generation is not equal for both runs. While it would be better to directly compare them based on the absolute amount of evaluations done at every point in time, this information was not available for the CMA-ES infused run. Nevertheless, the only difference between the two runs is their evolutionary framework. Thus while not ideal, wall clock time measured on the isolated cluster of eight compute nodes provides a relatively fair comparison.

And as can be seen on Figure 8.8, the CMA-ES infused approach resulted in a better performing crawler. Although the original approach outperformed the CMA-ES infused approach during the first 5 hours and 19 minutes, it did not improve upon its best performing crawler that was already evolved after 2 hours and 32 minutes. The most likely explanation being that it got stuck in a *local optimum* due to a lack of increasing genotypic (CPPN) complexity. This lack of genotypic complexity in turn was hypothesized to originate from the lack of *innovation protection*, which the CMA-ES infused approach aimed to mitigate.

The difference in average genotypic CPPN complexity over time is shown in Figure 8.9. This figure clearly shows that the CMA-ES infused approach evolved much more complex CPPN genomes. While one could argue that this may be due to the more aggressive structural mutation probabilities employed in the CMA-ES infused run, these same probabilities were also tested with the original CVT-MAP-Elites approach, which led to similar genotypic complexities as those of the CVT-MAP-Elites approach shown here (cfr. Section 5.3.1). This

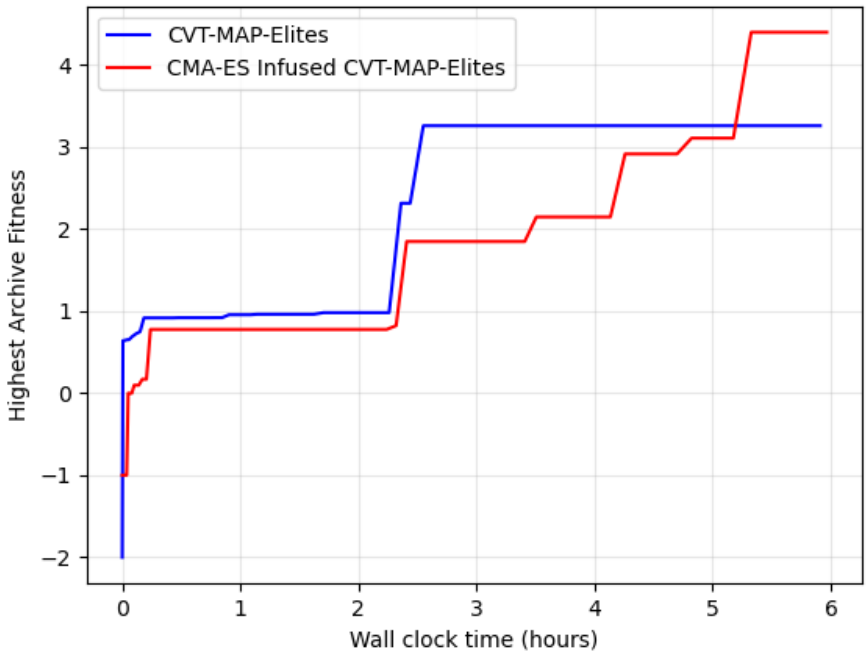
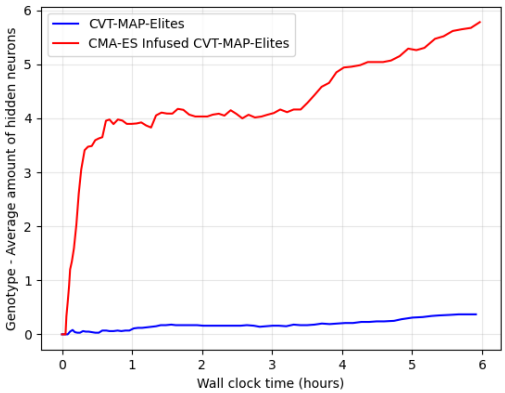
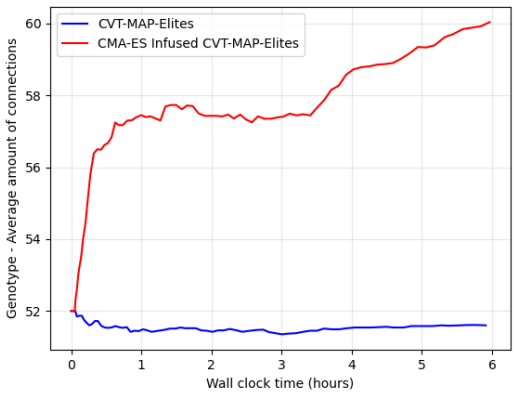


Figure 8.8: The highest fitness of an individual stored in the CVT-MAP-Elites archive of both runs, shown over the measured wall clock time. While the original CVT-MAP-Elites outperformed the CMA-ES infused approach during the first 5 hours and 19 minutes, it did not improve upon its best performing crawler that was already evolved after 2 hours and 32 minutes. In the end, the CMA-ES infused approach evolved the best performing crawler of both runs.



(a) The average amount of hidden neurons of the CPPN genomes stored in the CVT-MAP-Elites archive over the measured wall clock time.



(b) The average amount of connections of the CPPN genomes stored in the CVT-MAP-Elites archive over the measured wall clock time.

Figure 8.9: Comparison of the genotypic CPPN complexities of both the original CVT-MAP-Elites approach and the CMA-ES infused extension. As can be seen, the CMA-ES infused approach allows more complex CPPN genomes to arise. This is hypothesized to be due to the incorporation of an explicit innovation protection mechanism.

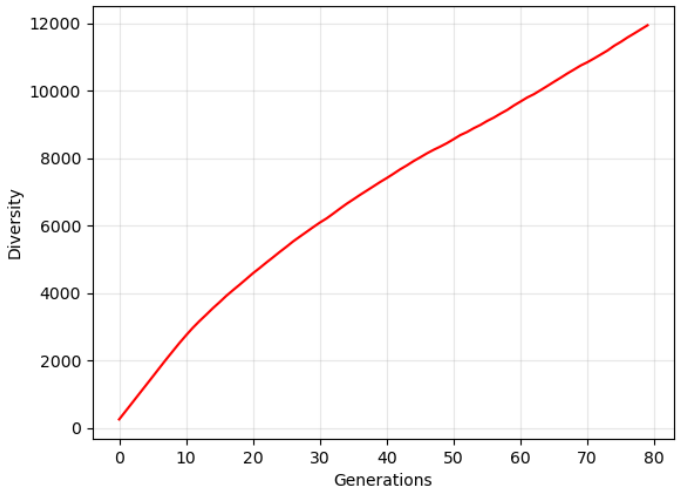
further substantiates our hypothesis concerning the "lack of innovation protection" of the CVT-MAP-Elites approach that arises when the underlying genetic encoding is a neural network such as the CPPN in our case. Nevertheless, given that only one run of each approach was done here (due to time constraints of the author), these results should only be seen as an indication of the potential strengths of the proposed CMA-ES infused extension. More experiments and tuning of the proposed CMA-ES infused approach are required, given the inherent stochasticity of evolutionary approaches like these.

Next to the incorporation of an explicit innovation protection mechanism, the CMA-ES infused approach also tried to mitigate an inefficiency concerning "the lack of information reuse". The key idea behind this was that each time some genome structure would be evaluated in the original approach, we now instead continued its stored CMA-ES instance. These CMA-ES instances were stored in an archive, named the *ES-Archive*. Figure 8.10 gives an

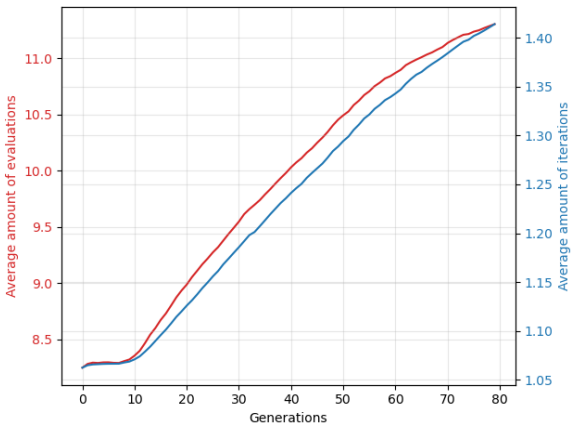
overview of some metrics regarding that archive that were recorded during the experiment. In contrary to the previous figures, these are now shown in terms of generations; given that we are no longer comparing the two approaches.

Figure 8.10a depicts the diversity of the ES-archive over time. This diversity is the number of distinct CMA-ES instances (or CPPN genome structures) stored in the ES-archive. As a consequence of the relatively high structural mutation probabilities used, this increases quite rapidly and results into a total of approximately 12000 distinct CMA-ES instances after 79 generations. While the average amount of evaluations and iterations of all CMA-ES instances stored in the ES-Archive is shown in Figure 8.10b, the maximum amounts over all CMA-ES instances in the ES-Archive is shown in Figure 8.10c. The main insight gained from this is that the CMA-ES infused approach indeed reuses previously available optimization information, given that it advances previously stored CMA-ES instances (otherwise the average and maximum amounts of evaluations and iterations of the CMA-ES instances would not increase).

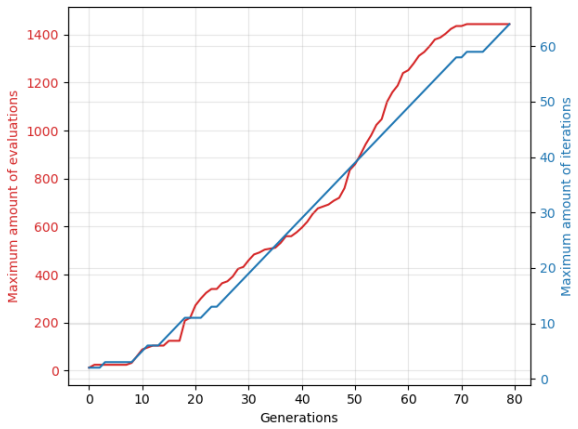
Lastly, we compare the phenotypes of the *fittest* crawlers evolved by the two approaches. These are schematically presented in Figure 8.11. While the average phenotypic controller complexities of the two runs in terms of the amount of hidden neurons remained similar, the fittest crawler of the CMA-ES infused run evolved approximately 4.5 times more connections compared to the fittest crawler evolved by the original CVT-MAP-Elites approach. This difference is due to the difference in genotypic complexity. As increasing genotypic complexity allows for increasingly complex output patterns, these patterns are likely to contain more variance. Given that the AESHN algorithm is used to create the neural network controllers, a higher variance allows more connections to be placed. As can be seen in the figure, the higher genotypic CPPN complexity of the CMA-ES infused approach also allowed a more



(a) The number of CMA-ES instances (diversity of CPPN genome structures) stored in the ES-Archive over time.



(b) The average amount of evaluations and iterations of the CMA-ES instances stored in the ES-Archive over time.



(c) The maximum amount of evaluations and iterations of an CMA-ES instance stored in the ES-Archive over time.

Figure 8.10: Overview of metrics over time regarding the ES-Archive of the CMA-ES infused approach. The main insight gained from these is that the CMA-ES infused approach indeed reuses previously available optimization information by advancing stored CMA-ES instances.

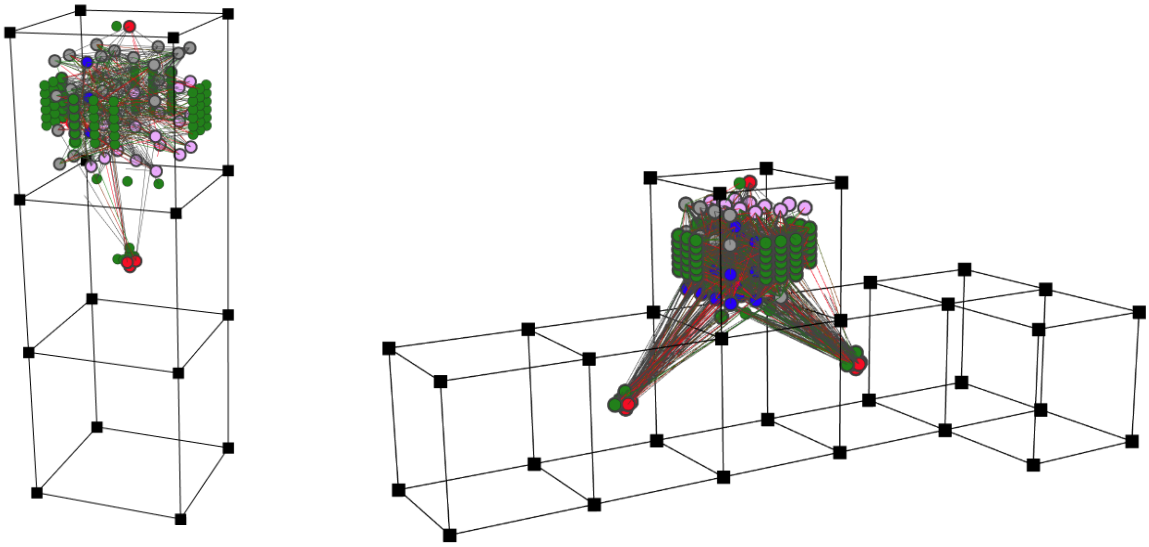
complex morphology to emerge.

While both the approaches were able to evolve crawlers that prolonged their lifespan by eating vegetation, neither evolved crawlers that did so in a manner that is deemed enough to conclude this experiment completely. Visual evaluation of the crawlers' behaviors showed that, while they do move towards a patch of vegetation and stick around it until the greater part of it is eaten, all of them eventually get stuck into one corner of the environment. Consequently, they die from starvation while there is still vegetation available in other areas of the environment. We also tried some visual evaluation runs wherein the patches of vegetation were moved from their original (fixed) location as used during training. Although this showed that some creatures just kept following the same path and in this case not reaching any vegetation, others showed adaptivity and still reached vegetation.

In hindsight, during the optimization process it would have been better to evaluate each creature within multiple independent trials, thereby randomizing the location of the vegetation in each trial such that creatures cannot just overfit towards following paths that always lead to some vegetation. The total fitness of a creature would then be set to the minimum of every such trial, consequently avoiding insignificant creatures to gain high fitness scores and misleading the optimization process. Although we were not able to test this ourselves due to a lack of time, we consider this the next step towards solving this environment.

Nevertheless, the results do showcase the potential of the proposed *One CPPN to Rule them All* (OCRA) approach to simultaneously evolve creature morphology and controller within this simplified virtual ecosystem. Additionally, the results obtained using the proposed CMA-ES extension of the CVT-MAP-Elites approach serve as an initial proof of concept of the extension, thereby providing a preliminary indication of its advantages. However, it

is important to note that this experiment (even combined with the OCRA specific experiments discussed in Chapter 5) is far from enough to count as a complete validation of both the *OCRA* and *CMA-ES infused CVT-MAP-Elites* approaches. As discussed in Chapter 9, many more experiments are necessary but are left for future work.



(a) Fittest crawler evolved by the CVT-MAP-Elites run after 2 hours and 32 minutes wall clock time. The controller contains 33 standard neurons (gray), 4 recurrent neurons (blue), 17 neuromodulatory neurons (pink) and a total of 623 connections. This crawler obtained a fitness of 3.261.

(b) Fittest crawler evolved by the CMA-ES infused CVT-MAP-Elites run after 5 hours and 19 minutes wall clock time. The controller contains 22 standard neurons (gray), 35 recurrent neurons (blue), 14 neuromodulatory neurons (pink) and a total of 2771 connections. This crawler obtained a fitness of 4.401.

Figure 8.11: Schematic visualizations of the best performing crawler using the CVT-MAP-Elites approach (a) and using the CMA-ES infused CVT-MAP-Elites approach (b). Due to the increased genotypic CPPN complexity of the CMA-ES infused approach, a higher phenotypic complexity was evolved. Fitness here is defined as the amount of *food-energy* replenished by the creature during its lifetime.

CHAPTER 9

CONCLUSION, IMPACT AND FUTURE WORK

As its title implies, the ultimate goal of this thesis was to evolve virtual creatures. Therefore, we identified two research objectives. One of which aimed to mitigate the current issues encountered when simultaneously evolving morphology and control; subsequently yielding a novel methodology towards the creation of such virtual creatures through geometrical patterns. The other research objective was concerned with producing a more comprehensive environment in which these creatures can be evolved.

In this final chapter we reflect on these research objectives. Thereby we give an overview of what has been done, discuss the potential impact of the work presented here and suggest possible directions for future work.

9.1 A Novel Approach Towards Complete Agent Evolution

Within the first research objective, we aimed to address the current limitations of simultaneously evolving agent morphology and control. These limitations were hypothesized by Cheney et al. [9] to originate from an effect called *"embodied cognition"*, in which an individual's body plan and brain have the incentive to specialize their behaviors to complement one another. This specialization makes improvements to either subsystem difficult without complementary changes in the other. Such complementary changes are highly unlikely in current evolutionary algorithms, and consequently the effect of embodied cognition establishes a significant impediment for them.

This thesis proposed to address this issue by completely representing agents using only a single genetic encoding. The key insight being that when evolving such a single genetic

encoding, the simultaneous evolution of both the morphology and the controller of the agent is fundamentally incorporated. However, this requires a powerful genetic encoding. Based on an exhaustive study of related work within this domain, we chose to employ the *"Compositional Pattern Producing Network"* (CPPN) as the genetic encoding. As its name suggests, CPPNs produce patterns within some geometrical space and their usage as a genetic encoding lies in the fact that we can interpret these patterns as a blueprint describing the agent. In literature, CPPNs have already been successfully used to separately encode functional agent morphologies and controllers, albeit never to simultaneously represent both. Consequently, this brought rise to the novel methodology proposed in this thesis, named *"One CPPN to Rule them All"* or *"OCRA"* in short. Within this novel methodology, the same CPPN is used to generate both morphology and controller defining patterns within the same geometrical space, thereby encoding the agent completely. The main hypothesis behind the approach being that the CPPN could be evolved in such a way that it learns an appropriate relationship between the morphology and controller defining patterns that it generates. This in turn causing mutations on the CPPN to naturally lead to complementary changes within both of these patterns and by extent in the morphology and controller that it represents.

To substantiate this hypothesis and to put the proposed methodology to the test, we carried out a series of experiments. These experiments started with a validation of the employed neuroevolution algorithm, named *Adaptive Evolvable-Substrate HyperNEAT* [100]. This was necessary in order to verify the minor modifications that we made to this algorithm. Afterwards, the morphological evolution was incorporated as well, thereby establishing the first experiment employing the complete OCRA approach. Without much trouble, this experiment showed the ability of the OCRA methodology to produce walking creatures.

The next step was then to integrate the usage of sensors within these evolved creatures.

While the original idea was that these sensors could be evolved as part of the block-based morphology of the creatures, thereby allowing the CPPN genome to express the placement of *"sensor blocks"* through a distinct pattern, a series of experiments showed that this is not as straightforward as initially hoped. Nevertheless, the failed experiments proved to be useful in the insights they provided and indicated the need for a complexity reduction. Next to a simplification of the environment that better allowed to validate the actual sensor usage of creatures, we eliminated the need for the OCRA approach to evolve an appropriate placing for these sensors. Therefore, we fixed the sensors to the *brain block* of the creatures, thereby only allowing other blocks to be evolved beneath this brain block. Semantically, this introduced the shift from *creatures* to *crawlers*. In the end, this resulted into the OCRA approach successfully evolving complete crawlers, capable of moving towards target locations identified by their fixed set of sensors.

The final series of experiments then evaluated these crawlers in a simplified version of the virtual ecosystem discussed below. Therein, each evolved crawler was individually evaluated and had to survive as long as possible through eating vegetation. Although the OCRA methodology was capable of evolving a set of crawlers that exhibited such survivalistic behavior, this experiment highlighted some issues with the employed evolutionary framework, namely MAP-Elites [41]. One such issue is *"the lack of innovation protection"* that arises when the underlying genetic encoding is a neural network such as the CPPN in our case. Additionally, we hypothesized that there was an inefficiency concerning the *"lack of optimization information reuse"*. To address these issues, we proposed an extension of the MAP-Elites algorithm, named *"CMA-ES infused MAP-Elites"*. The initial results of this extension substantiated our hypotheses concerning its advantages and served as a preliminary indication of its potential.

9.1.1 Impact

In line with our aspirations, we have introduced a novel methodology towards simultaneous evolution of agent brain and morphology. Such a methodology serves purpose in both the evolutionary robotics and artificial life domains. Within the evolutionary robotics domain, evolving complete robots can be seen as the holy grail, as it would allow to exploit evolution's unlimited creative potential and reduce the amount of human efforts necessary for the design of robots. However, current works are still hindered in this scenario. Our methodology addresses the issues related to the effect of embodied cognition and thereby sheds light on the path towards that holy grail. Within the artificial life domain, the higher level of phenotypic freedom granted by our methodology allows to investigate a whole new range of possible creature interactions and behaviors. This will be further discussed in Section 9.2.

Additionally, an extension to the popular MAP-Elites framework was proposed. This extension allows to mitigate the issues that emerge when the underlying genetic encoding employed is a neural network. Outside of its utility within the OCRA methodology, this extension broadens the effectiveness of the MAP-Elites algorithm to other scenarios as well.

9.1.2 Future Work

The experiments done within this thesis provide a solid proof of concept of the proposed methodology. However, much more experimentation is required in order to truly delineate both its advantages and possible drawbacks. Therein, comparative benchmarks to existing systems play an important aspect. This statement applies to both the OCRA methodology and the proposed CMA-ES infused MAP-Elites extension. Next to continued validation of the presented approaches, we indicate some other possible directions for future work. Many of these are regarding the underlying genetic encoding, the CPPN, given its central position within the proposed methodology.

A fruitful pursuit would be to perform a detailed study of the effects of CPPN mutations on both the morphology and controller defining patterns that it generates. This would allow to validate and illuminate the CPPNs ability to learn an appropriate relationship between the morphology and controller that it encodes.

Furthermore, an important concept that was not investigated during this thesis is the *geometric seeding* of the CPPN [62]. Such seeding would incorporate a set of user defined hidden neurons within the initial population of CPPN genomes, such that these are biased towards certain types of structures. This is important as it provides a mechanism for emulating key biases in the natural world, such as symmetrical morphologies, brain modularity and cortical columns found in the human brains [95].

Another possible direction is the introduction of a dynamic mutation rate. Given that mutation rate somewhat relates to the *learning rate* parameter [124] employed in the artificial neural network domain, we believe that similar techniques could be applied here. Moreover, the graph based structure of the CPPN opens up the possibility of applying *Graph Neural Networks* (GNNs) [125] to them. Graph Neural Networks are neural models that capture the dependence of graphs via message passing between the nodes of the graphs. Some of their use cases, including missing link and node detection, give rise to the idea that they could potentially be used as a mutation operator.

9.2 Towards a More Comprehensive Virtual Ecosystem

The second research objective of this thesis concerned itself with addressing the limitations of the virtual worlds in which evolution has been studied to date. These limitations were identified by T. Taylor in [21] and include (1) a lack of complex dynamics within the environ-

ment, (2) the restrictiveness of ecological interactions (both between creatures and between the creatures and their environment) and (3) the lack of mixing genetic material between creatures. Such limitations are believed to constitute a significant part of the reason why current work is yet to succeed in reproducing the long-term evolutionary dynamics as observed in our biological world. Although many other elements play a role in the elusiveness of obtaining such an open-ended evolutionary process, the focus of study here were the problems inherent to the system or environment in which evolution takes place.

Drawing inspiration from a fundamental work within this domain, named PolyWorld [25], we built a more comprehensive environment that fundamentally incorporates the missing components of current systems described above. This environment was chosen to be a virtual ecosystem. As we try to approximate the remarkable capabilities of evolution in our biological world, a (simplified) virtual twin of our world accordingly forms an attractive environment. Within such an environment, the creatures are given the individual goal of surviving through collecting energy available in the environment. Collectively, their goal is to sustain their numbers through their own mating behaviors, thereby no longer requiring additional *artificial* creations to maintain a minimum amount of creatures.

To build this environment, we took a bottom-up approach that defines the whole virtual ecosystem (and its inhabitants) by means of modular building blocks. The key idea being that such a modular world allows to more easily introduce complex environmental dynamics, such as ever-changing landscapes due to blocks being moved around or stochastically changing the properties of the inserted blocks.

The restrictiveness of interactions in many works originates from the usage of a predefined or "hard-coded" set of actions for the creatures. In other words, a limited range of actions

limits the range of interactions. The need for such a predefined set of actions was alleviated by relying on the phenotypic freedom of creatures that methods such as the proposed OCRA approach grant the evolutionary process. This phenotypic freedom induced by the simultaneous evolution of both creature morphology and controller introduces a theoretically unlimited range of interactions between creatures. The key insight behind this being that the evolution of creatures can now lead to creatures discovering these actions and interactions themselves. Furthermore, combining this phenotypic freedom with a modular virtual ecosystem composed out of mutable building blocks naturally opens up a whole new range of interactions between the creature and its environment. One type of environment block, the *randomly defined block*, was specifically designated for this purpose.

Nevertheless, some basic actions are initially predefined to lower the complexity threshold of creatures surviving in the virtual ecosystem. These included *eating*, *drinking*, *attacking* and *mating*. While eating and drinking had trivial meanings, the attacking and mating actions allowed us to naturally introduce two important (inter-creature) interaction related concepts: *predation* and *sexual reproduction*. Sexual reproduction forms the main strategy for letting creatures reach their collective goal, next to allowing a "vertical" mixture of genetic material (offspring carries on mutated and crossed over genes of its parents).

Besides the virtual creatures, the presented virtual ecosystem hosts another type of virtual organism that is subject to evolution, although at a much higher level of complexity. This organism serves as the main food source of the virtual creatures and is portrayed as vegetation. The main interaction between these virtual organisms being that creatures can eat vegetation. Moreover, an additional set of interactions originated from the ability of creatures to move vegetation around. These interactions, combined with the genetic variation induced by evolution of both of these virtual organisms, served as an additional stimulus

for the evolution of both of these organisms.

Although some experiments were carried out, all of them took place in simplified versions of the proposed virtual ecosystem. The main reason for these simplifications being the novelty of the proposed OCRA approach. Given its novelty, we could not expect it to immediately take on the complexity of the complete virtual ecosystem and lead to "satisfying" results. Nevertheless, the complete virtual ecosystem was fully implemented and stands ready for further experimentation[1].

9.2.1 Impact

In line with our aspirations, we have introduced a more comprehensive virtual world in which artificial evolution can be studied. Our system fundamentally incorporates important stimuli for a more open-ended evolutionary dynamic, involving long-term intrinsic drives for increased diversity and complexity which are believed to be lacking in the systems available to date. Consequently, it allows us to take the next step towards unshackling the full potential of virtual evolution. Given its biologically inspired setup, inherent scalability and extensibility, the presented environment can serve as a test bed for many future evolutionary works within the artificial life domain. It is our hope that some of these works will include the proposed OCRA approach, as we believe that this environment can unlock its true potential.

9.2.2 Future Work

As indicated throughout the discussion of the virtual creatures in Section 7.2, an important future work is to improve the manner in which creatures can perceive both their environment and each other. Currently, this is done through fixed raycast sensors. Better would be to allow for visual observations. Although processing such visual observations raises complexity

1. The virtual ecosystem is available at: `https://github.ugent.be/dmarzoug/thesis`

(given that they encapsulate much more information), they are considerably more generic and conform better to our "open-ended" design. An additional advantage of visual observations is that they could allow for an explicit communication mechanism between creatures. This could for instance be done through color changing morphological blocks, as witnessed in PolyWorld [25].

Besides observations, we believe that it would be interesting to incorporate the notion of two *"sexes"* as seen in our biological world. Each creature would then be (stochastically) designated to one and the carrying of offspring would be limited to one such sex. Granted our current mechanism of sexual reproduction that incorporates a *gestation period*, this in turn could give rise to specialized, "sex-dependent" behaviors.

Furthermore, the existence of the inhabiting organisms could be expanded towards the aquatic world. Whereas current life is limited to *earth* blocks, the *water* blocks could provide an interesting habitat as well. Given that such an aquatic world is characterized by a different set of physics, this has the potential to yield an increased variety of evolved virtual creatures. However, such fluid dynamics are yet to be incorporated within the system.

The modularity and extensibility of the proposed virtual ecosystem, by themselves, also allow for many possible directions of future work. Examples include (1) the introduction of different environmental building blocks, (2) introducing more complex landscapes such as mountain ranges (evolving climbing creatures) and (3) temporal modifications to the properties of the environmental blocks to simulate seasons.

Successful simulations, wherein the creatures are capable of maintaining their own numbers purely through sexual reproduction, could be check-pointed and used as a starting point to

investigate the behavioral effects of *"external shocks"*. Such external shocks could include (what we like to call) a *"Thanos-snap"*, i.e. randomly eliminating half the population of creatures, or introducing a contagious virus.

Perhaps in a more fanciful and visionary vein, we could evolve the complete virtual ecosystem itself as well by encoding it through a *Compositional Pattern Producing Network*. Granted that the properties of the geometric patterns that it produces are not only interesting to represent virtual creatures, this poses as an appealing road to follow. Instead of *"Evolving Virtual Creatures through Geometrical Patterns"*, we would be *"Evolving Virtual Worlds through Geometrical Patterns"*.

BIBLIOGRAPHY

[1] Nick Cheney, Josh Bongard, Vytas SunSpiral, and Hod Lipson. Scalable co-optimization of morphology and control in embodied machines. *J. R. Soc. Interface*, 15(143), June 2018.

[2] Thomas Geijtenbeek, Michiel van de Panne, and A Frank van der Stappen. Flexible muscle-based locomotion for bipedal creatures, 2013.

[3] Antoine Cully, Jeff Clune, and Jean-Baptiste Mouret. Robots that can adapt like natural animals. *CoRR*, abs/1407.3501, 2014.

[4] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-End training of deep visuomotor policies. April 2015.

[5] Steve Collins, Andy Ruina, Russ Tedrake, and Martijn Wisse. Efficient bipedal robots based on passive-dynamic walkers. *Science*, 307(5712):1082–1085, February 2005.

[6] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. *SIGEVOlution*, 7(1):11–23, August 2014.

[7] J. Auerbach and J. Bongard. Environmental influence on the evolution of morphological complexity in machines. *PLoS Computational Biology*, 10, 2014.

[8] T Geijtenbeek and N Pronost. Interactive character animation using simulated physics: A State-of-the-Art review, 2012.

[9] Nicholas Cheney, Josh Bongard, Vytas Sunspiral, and Hod Lipson. On the difficulty of Co-Optimizing morphology and control in evolved virtual creatures, 2016.

[10] B Weel, E Crosato, J Heinerman, E Haasdijk, and A E Eiben. A robotic ecosystem with evolvable minds and bodies. In *2014 IEEE International Conference on Evolvable Systems*, pages 165–172, December 2014.

[11] Allan Zhao, Jie Xu, Mina Konaković-Luković, Josephine Hughes, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. RoboGrammar: graph grammar for terrain-optimized robot design. *ACM Trans. Graph.*, 39(6):1–16, November 2020.

[12] Jørgen Nordmoen, Frank Veenstra, Kai Olav Ellefsen, and Kyrre Glette. MAP-Elites enables powerful stepping stones and diversity for modular robotics, 2020.

[13] Angelo Cangelosi, Josh Bongard, Martin H Fischer, and Stefano Nolfi. Embodied intelligence. In Janusz Kacprzyk and Witold Pedrycz, editors, *Springer Handbook of Computational Intelligence*, pages 697–714. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[14] Kenneth O Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genet. Program. Evolvable Mach.*, 8(2):131–162, June 2007.

[15] Sam Kriegman, Douglas Blackiston, Michael Levin, and Josh Bongard. A scalable pipeline for designing reconfigurable organisms. *Proc. Natl. Acad. Sci. U. S. A.*, 117(4):1853–1859, January 2020.

[16] J Clune, K O Stanley, R T Pennock, and C Ofria. On the performance of indirect encoding across the continuum of regularity. *IEEE Trans. Evol. Comput.*, 15(3):346–367, June 2011.

[17] S Risi and K O Stanley. A unified approach to evolving plasticity and neural geometry. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, June 2012.

[18] Collin Cappelle and Josh Bongard. Embodied embeddings for hyperneat. *Artificial Life Conference Proceedings*, (30):461–468, 2018.

[19] Pablo Reyes and María-José Escobar. Neuroevolutive algorithms for learning gaits in legged robots. *IEEE Access*, 7:142406–142420, 2019.

[20] Sam Kriegman. Why virtual creatures matter. *Nature Machine Intelligence*, 1:492–492, 10 2019.

[21] Tim Taylor. Evolution in virtual worlds. *CoRR*, abs/1710.06055, 2017.

[22] T. Ray. An approach to the synthesis of life. 1991.

[23] Thomas Miconi. Evosphere: Evolutionary dynamics in a population of fighting virtual creatures. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 3066–3073, 2008.

[24] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011.

[25] Larry Yaeger. Computational genetics, physiology, metabolism, neural systems, learning, vision, and behavior or polyworld: Life in a new context. 03 1995.

[26] Peter T. Hraber, Terry Jones, and Stephanie Forrest. The Ecology of Echo. *Artificial Life*, 3(3):165–190, 07 1997.

[27] Kenneth De Jong. Evolutionary computation: A unified approach. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, GECCO '16 Companion, pages 185–199, New York, NY, USA, July 2016. Association for Computing Machinery.

[28] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.

[29] Konstantinos Chatzilygeroudis, Antoine Cully, Vassilis Vassiliades, and Jean-Baptiste Mouret. Quality-Diversity optimization: a novel branch of stochastic optimization. *ArXiv*, abs/2012.04322, 2020.

[30] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 312–317, 1996.

[31] Ingolfson. CMA-ES - Wikipedia , The Free Encyclopedia, 2017.

[32] Stefano Nolfi and Dario Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, 2000.

[33] Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, January 2019.

[34] H. Spencer. *The Principles of Biology*. Number v. 1 in Nineteenth Century Collections Online (NCCO): Science, Technology, and Medicine: 1780-1925. Williams and Norgate, 1864.

[35] Joel Lehman and Kenneth Stanley. Revising the evolutionary computation abstraction. pages 103–110, 01 2010.

[36] Joel Lehman and Kenneth O. Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, page 211–218, New York, NY, USA, 2011. Association for Computing Machinery.

[37] Sebastian Risi, Sandy D. Vanderbleek, Charles E. Hughes, and Kenneth O. Stanley. How novelty search escapes the deceptive trap of learning to learn. In *Proceedings of*

the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO '09, page 153–160, New York, NY, USA, 2009. Association for Computing Machinery.

[38] Jorge C. Gomes, Paulo Urbano, and Anders Lyhne Christensen. Evolution of swarm robotics systems with novelty search. *CoRR*, abs/1304.3362, 2013.

[39] Noe Casas. Genetic algorithms for multimodal optimization: a review. *CoRR*, abs/1508.05342, 2015.

[40] Jean-Baptiste Mouret and Glenn Maguire. Quality diversity for multi-task optimization. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, page 121–129, New York, NY, USA, 2020. Association for Computing Machinery.

[41] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites, 2015.

[42] Danesh Tarapore, Jeff Clune, Antoine Cully, and Jean-Baptiste Mouret. How do different encodings influence the performance of the map-elites algorithm? In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, page 173–180, New York, NY, USA, 2016. Association for Computing Machinery.

[43] Konstantinos I. Chatzilygeroudis, Vassilis Vassiliades, and Jean-Baptiste Mouret. Reset-free trial-and-error learning for data-efficient robot damage recovery. *CoRR*, abs/1610.04213, 2016.

[44] Justin K. Pugh, L. B. Soros, Paul A. Szerlip, and Kenneth O. Stanley. Confronting the challenge of quality diversity. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, page 967–974, New York, NY, USA, 2015. Association for Computing Machinery.

[45] Vassilis Vassiliades, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. Using centroidal voronoi tessellations to scale up the multidimensional archive of phenotypic elites algorithm. *IEEE Transactions on Evolutionary Computation*, 22:623–630, 2018.

[46] Q. Du, V. Faber, and M. Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *SIAM Rev.*, 41:637–676, 1999.

[47] Nikolaus Hansen. The cma evolution strategy: A tutorial, 2016.

[48] Sentewolf. CMA-ES - Wikipedia , The Free Encyclopedia, 2020.

[49] Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Trans. Graph.*, 32(6), November 2013.

[50] V. Heidrich-Meisner and C. Igel. Neuroevolution strategies for episodic reinforcement learning. *J. Algorithms*, 64:152–168, 2009.

[51] Nils Müller and Tobias Glasmachers. Challenges in high-dimensional reinforcement learning with evolution strategies, 2018.

[52] Youhei Akimoto, Anne Auger, and Nikolaus Hansen. Comparison-based natural gradient optimization in high dimension. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, page 373–380, New York, NY, USA, 2014. Association for Computing Machinery.

[53] Ilya Loshchilov, Tobias Glasmachers, and Hans-Georg Beyer. Limited-memory matrix adaptation for large scale black-box optimization. *CoRR*, abs/1705.06693, 2017.

[54] Ilya Loshchilov. A computationally efficient limited memory CMA-ES for large scale optimization. *CoRR*, abs/1404.5520, 2014.

[55] Raymond Ros and N. Hansen. A simple modification in cma-es achieving linear time and space complexity. In *PPSN*, 2008.

[56] John Von Neumann and Arthur W. Burks. *Theory of Self-Reproducing Automata*. University of Illinois Press, USA, 1966.

[57] Tim Taylor. *From Artificial Evolution to Artificial Life*. PhD thesis, 06 1999.

[58] Nils Aall Barricelli. Numerical testing of evolution theories. *Acta Biotheoretica*, 16(1-2):69–98, 1962.

[59] N. A. Barricelli. Numerical testing of evolution theories. *Acta Biotheoretica*, 16:99–126, 1962.

[60] Michael Conrad and H.H. Pattee. Evolution experiments with an artificial ecosystem. *Journal of Theoretical Biology*, 28(3):393–409, 1970.

[61] Karl Sims. Evolving 3D morphology and behavior by competition. *Artif. Life*, 1(4):353–372, July 1994.

[62] S. Risi. Towards evolving more brain-like artificial neural networks. 2012.

[63] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

[64] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, 2002.

[65] Oliver J. Coleman, Alan D. Blair, and Jeff Clune. Automated generation of environments to test the general learning capabilities of ai agents. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, page 161–168, New York, NY, USA, 2014. Association for Computing Machinery.

[66] Maciej Komosiński and Szymon Ulatowski. Framsticks: Towards a simulation of a Nature-Like world, creatures and evolution. In *Advances in Artificial Life*, pages 261–265. Springer Berlin Heidelberg, 1999.

[67] Daniel Richards and Martyn Amos. Evolving morphologies with CPPN-NEAT and a dynamic substrate. *Artificial Life Conference Proceedings*, 26:255–262, July 2014.

[68] Lichun Cao and ZhiMin. An overview of deep reinforcement learning, 2019.

[69] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. September 2015.

[70] Frédéric Gruau. Automatic definition of modular neural networks. *Adapt. Behav.*, 3(2):151–183, September 1994.

[71] Xin Yao. A review of evolutionary artificial neural networks, 1993.

[72] Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evol. Intell.*, 1(1):47–62, March 2008.

[73] Andrea Soltoggio, Kenneth O Stanley, and Sebastian Risi. Born to learn: The inspiration, progress, and future of evolved plastic artificial neural networks. *Neural Netw.*, 108:48–67, December 2018.

[74] D. Dasgupta and D. R. McGregor. Designing application-specific neural networks using the structured genetic algorithm. *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 87–96, 1992.

[75] João Pujol and Riccardo Poli. Evolving the topology and the weights of neural networks using a dual representation. *Appl. Intell.*, 8:73–84, 01 1998.

[76] Josh C. Bongard and Rolf Pfeifer. Evolving complete agents using artificial ontogeny. In Fumio Hara and Rolf Pfeifer, editors, *Morpho-functional Machines: The New Species*, pages 237–258, Tokyo, 2003. Springer Japan.

[77] Frédéric Gruau. Genetic synthesis of modular neural networks. In *Proceedings of the 5th International Conference on Genetic Algorithms*, page 318–325, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[78] David E Moriarty and Risto Miikkulainen. Evolving obstacle avoidance behavior in a robot arm. In *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 468–475, 1996.

[79] H Lipson and J B Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, August 2000.

[80] G S Hornby, S Takamura, J Yokono, O Hanagata, T Yamamoto, and M Fujita. Evolving robust gaits with AIBO. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 3, pages 3040–3045 vol.3, April 2000.

[81] T. Aaltonen, J. Adelman, T. Akimoto, M. G. Albrow, B. Álvarez González, S. Amerio, D. Amidei, A. Anastassov, A. Annovi, J. Antos, and et al. Measurement of the top-quark mass with dilepton events selected using neuroevolution at cdf. *Physical Review Letters*, 102(15), Apr 2009.

[82] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, 2011.

[83] Jeff Clune, Jean-Baptiste Mouret, and Hod Lipson. The evolutionary origins of mod-

ularity. *Proceedings of the Royal Society B: Biological Sciences*, 280(1755):20122863, Mar 2013.

[84] Joost Huizinga, Jeff Clune, and Jean-Baptiste Mouret. Evolving neural networks that are both modular and regular: Hyperneat plus the connection cost technique. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, page 697–704, New York, NY, USA, 2014. Association for Computing Machinery.

[85] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. March 2017.

[86] Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. Natural evolution strategies, 2008.

[87] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *CoRR*, abs/1712.06567, 2017.

[88] David J. Montana and Lawrence Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'89, page 762–767, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[89] D. Dasgupta and D.R. McGregor. Designing application-specific neural networks using the structured genetic algorithm. In *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 87–96, 1992.

[90] Kenneth Stanley and Risto Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. 04 2002.

[91] Kenneth Stanley, David D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15:185–212, 02 2009.

[92] Sebastian Risi and Kenneth Stanley. Enhancing es-hyperneat to evolve more complex regular neural networks. pages 1539–1546, 01 2011.

[93] S. Risi and K. Stanley. An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. *Artificial Life*, 18:331–363, 2012.

[94] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Inf.*, 4(1):1–9, March 1974.

[95] Olaf Sporns. Network analysis, complexity, and brain function. *Complexity*, 8(1):56–60, 2002.

[96] D Floreano and J Urzelai. Evolutionary robots with on-line self-organization and behavioral fitness. *Neural Networks*, 13(4):431–443, 2000.

[97] Yael Niv, Daphna Joel, Isaac Meilijson, and Eytan Ruppin. Evolution of reinforcement learning in uncertain environments: A simple explanation for complex foraging behaviors. *Adapt Behav*, 10(1):5–24, April 2002.

[98] A. Soltoggio, J. Bullinaria, C. Mattiussi, Peter Dürr, and D. Floreano. Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. In *ALIFE*, 2008.

[99] S. Risi and K. Stanley. Indirectly encoding neural plasticity as a pattern of local rules. In *SAB*, 2010.

[100] S. Risi and K. O. Stanley. A unified approach to evolving plasticity and neural geometry. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2012.

[101] Angelo Cangelosi, Josh Bongard, Martin Fischer, and Stefano Nolfi. Embodied intelligence. *Springer Handbook of Computational Intelligence*, pages 697–714, 01 2015.

[102] Jia Han, Gordon Waddington, Roger Adams, Judith Anson, and Yu Liu. Assessing proprioception: A critical review of methods. *Journal of Sport and Health Science*, 5(1):80–90, 2016.

[103] Apoorva Kelkar and John D. Medaglia. *Evidence of Brain Modularity*, pages 1–10. Springer International Publishing, Cham, 2018.

[104] David B. D'Ambrosio, J. Lehman, S. Risi, and K. Stanley. Evolving policy geometry for scalable multiagent learning. In *AAMAS*, 2010.

[105] Sebastian Risi and Kenneth O. Stanley. Confronting the challenge of learning a flexible neural controller for a diversity of morphologies. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, page 255–262, New York, NY, USA, 2013. Association for Computing Machinery.

[106] Gregory Morse, Sebastian Risi, Charles Snyder, and Kenneth Stanley. Single-unit pattern generators for quadruped locomotion. pages 719–726, 07 2013.

[107] Manon Flageat and Antoine Cully. Fast and stable map-elites in noisy domains using deep grids. *CoRR*, abs/2006.14253, 2020.

[108] Jinpeng Liu and K. Tang. Scaling up covariance matrix adaptation evolution strategy using cooperative coevolution. In *IDEAL*, 2013.

[109] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.

[110] OpenAI. Openai gym - cartpole-v0.

[111] A. Moore. Efficient memory-based learning for robot control. 1990.

[112] OpenAI. Openai gym - mountaincar-v0.

[113] Unity Technologies. Unity.

[114] Unity Technologies. Unity ml-agents toolkit.

[115] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. September 2018.

[116] SP Lloyd. Least square quantization in pcm. bell telephone laboratories paper. published in journal much later: Lloyd, sp: Least squares quantization in pcm. *IEEE Trans. Inform. Theor.(1957/1982)*, 18, 1957.

[117] Mojang. Minecraft.

[118] Anssi Kanervisto, Janne Karttunen, and Ville Hautamäki. Playing minecraft with behavioural cloning. *CoRR*, abs/2005.03374, 2020.

[119] Minecraft open-endedness challenge.

[120] Ada King. Unity asset store - free trees.

[121] Yuxi Li. Deep reinforcement learning: An overview. *CoRR*, abs/1701.07274, 2017.

[122] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[123] Yuri Burda, Harrison Edwards, Deepak Pathak, Amos J. Storkey, Trevor Darrell, and Alexei A. Efros. Large-scale study of curiosity-driven learning. *CoRR*, abs/1808.04355, 2018.

[124] Leslie N. Smith. No more pesky learning rate guessing games. *CoRR*, abs/1506.01186, 2015.

[125] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018.

[126] Nikolaus Hansen. The CMA evolution strategy: A tutorial. April 2016.

[127] Céline Neutens, Bart de Dobbelaer, Peter Claes, and Dominique Adriaens. Prehensile and non-prehensile tails among syngnathid fishes: what's the difference?, 2017.

[128] Wen Yang, Irene H Chen, Bernd Gludovatz, Elizabeth A Zimmermann, Robert O Ritchie, and Marc A Meyers. Natural flexible dermal armor. *Adv. Mater.*, 25(1):31–48, January 2013.

[129] Hans M Peters. Beiträge zur ökologischen physiologie des seepferdes (hippocampus brevirostris). *Z. Vgl. Physiol.*, 33(3):207–265, May 1951.

[130] Wen Yang, Steven E Naleway, Michael M Porter, Marc A Meyers, and Joanna McKittrick. The armored carapace of the boxfish. *Acta Biomater.*, 23:1–10, September 2015.

[131] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 2015. PMLR.

[132] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning, 2015.

[133] Céline Neutens. *Transforming tails into tools : from evolutionary morphology of prehensile tails in syngnathid fishes to exploring bio-inspiration potentials*. PhD thesis, Ghent University, 2016.

[134] Steven E Naleway, Jennifer R A Taylor, Michael M Porter, Marc A Meyers, and Joanna McKittrick. Structure and mechanical properties of selected protective systems in marine organisms. *Materials Science and Engineering: C*, 59:1143–1167, 2016.

[135] Gabriel Urbain, Jonas Degrave, Benonie Carette, Joni Dambre, and Francis Wyffels. Morphological properties of Mass–Spring networks for optimal locomotion learning, 2017.

[136] Mihoko Otake, Yoshiharu Kagami, Masayuki Inaba, and Hirochika Inoue. Motion design of a starfish-shaped gel robot made of electro-active polymer gel. *Rob. Auton. Syst.*, 40(2):185–191, August 2002.

[137] Sangbae Kim, Cecilia Laschi, and Barry Trimmer. Soft robotics: a bioinspired evolution in robotics. *Trends Biotechnol.*, 31(5):287–294, May 2013.

[138] You Hao Li, Qujiang Lei, Chaopeng Cheng, Gong Zhang, Zheng Xu, and others. A review: machine learning on robotic grasping. March 2019.

[139] M Manti, V Cacucciolo, and M Cianchetti. Stiffening in soft robotics: A review of the state of the art. *IEEE Robot. Autom. Mag.*, 23(3):93–106, September 2016.

[140] Daniela Rus and Michael T Tolley. Design, fabrication and control of soft robots. *Nature*, 521(7553):467–475, May 2015.

[141] Irene H Chen, James H Kiang, Victor Correa, Maria I Lopez, Po-Yu Chen, Joanna McKittrick, and Marc A Meyers. Armadillo armor: mechanical testing and microstructural evaluation. *J. Mech. Behav. Biomed. Mater.*, 4(5):713–722, July 2011.

[142] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 2016. PMLR.

[143] N Hansen and A Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 312–317, 1996.

[144] Brecht Willems, Jonas Degrave, Joni Dambre, and Francis Wyffels. Quadruped robots benefit from compliant leg designs. pages 3091–3091, Vancouver, 2017. IEEE.

[145] N Hansen and A Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 312–317, May 1996.

[146] Adam A Stokes, Robert F Shepherd, Stephen A Morin, Filip Ilievski, and George M Whitesides. A hybrid combining hard and soft robots. *Soft Robotics*, 1(1):70–74, March 2014.

[147] B S Homberg, R K Katzschmann, M R Dogar, and D Rus. Haptic identification of objects using a modular soft robotic gripper. In *2015 IEEE/RSJ International*

Conference on Intelligent Robots and Systems (IROS), pages 1698–1705, September 2015.

[148] Sebastian Risi and Kenneth O Stanley. An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. *Artif. Life*, 18(4):331–363, August 2012.

[149] Michiel Hermans, Benjamin Schrauwen, Peter Bienstman, and Joni Dambre. Automated design of complex dynamic systems. *PLoS One*, 9(1):e86696, January 2014.

[150] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. March 2018.

[151] Michael W Hannan and Ian D Walker. Kinematics and the implementation of an elephant's trunk manipulator and other continuum style robots. *J. Robot. Syst.*, 20(2):45–63, February 2003.

[152] M Ciocarlie and P Allen. Data-driven optimization for underactuated robotic hands. In *2010 IEEE International Conference on Robotics and Automation*, pages 1292–1299, May 2010.

[153] Deepak Trivedi, Christopher D Rahn, William M Kier, and Ian D Walker. Soft robotics: Biological inspiration, state of the art, and future research. *Appl. Bionics Biomech.*, 5(3):99–117, December 2008.

[154] A L Rosenberger. Tale of tails: parallelism and prehensility. *Am. J. Phys. Anthropol.*, 60(1):103–107, January 1983.

[155] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. July 2017.

[156] Hermann Weber. Beiträge zur bewegungsphysiologie der Hippocampus-Arten. *Z. Vgl. Physiol.*, 5(1):1–36, February 1927.

[157] L Wang and F Iida. Deformation in Soft-Matter robotics: A categorization and quantitative characterization. *IEEE Robot. Autom. Mag.*, 22(3):125–139, September 2015.

[158] Jingzhou Yang, Esteban Peña Pitarch, Jason Potratz, Steven Beck, and Karim Abdel-Malek. Synthesis and analysis of a flexible elephant trunk robot. *Adv. Robot.*, 20(6):631–659, January 2006.

[159] Jonas Degrave, Michaël Burm, Pieter-Jan Kindermans, Joni Dambre, and Francis Wyffels. Transfer learning of gaits on a quadrupedal robot. *Adapt. Behav.*, 23(2):69–82, April 2015.

[160] Lael U Odhner, Leif P Jentoft, Mark R Claffee, Nicholas Corson, Yaroslav Tenzer, Raymond R Ma, Martin Buehler, Robert Kohout, Robert D Howe, and Aaron M Dollar. A compliant, underactuated hand for robust manipulation. *Int. J. Rob. Res.*, 33(5):736–752, 2014.

[161] Ross M McKenzie, Thomas W Barraclough, and Adam A Stokes. Integrating soft robotics with ROS - a hybrid pick and place arm. February 2017.

[162] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *AAAI*, 32(1), April 2018.

[163] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Real-time neuroevolution in the nero video game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–668, 2005.

[164] D. Rumelhart, Geoffrey E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[165] B.T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[166] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks : the official journal of the International Neural Network Society*, 12(1):145—151, January 1999.

[167] Yann Lecun, Leon Bottou, Y. Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278 – 2324, 12 1998.

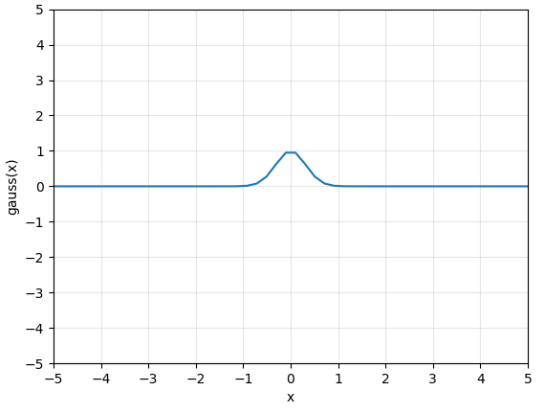
[168] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[169] Jeffrey Heaton. Ian goodfellow, yoshua bengio, and aaron courville: Deep learning: The mit press, 2016, 800 pp, isbn: 0262035618. *Genetic Programming and Evolvable Machines*, 19, 10 2017.

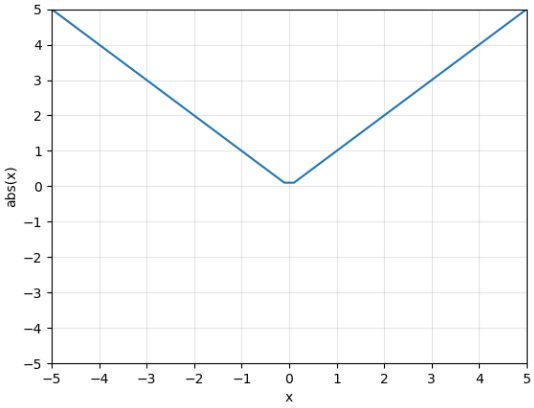
[170] Hunter Heidenreich. Neat: An awesome approach to neuroevolution, Jan 2019.

APPENDIX A

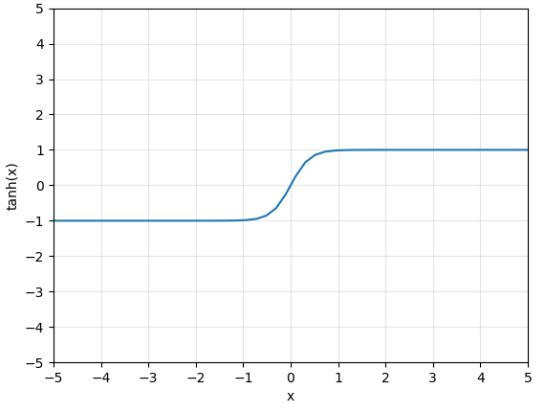
CPPN ACTIVATION FUNCTIONS



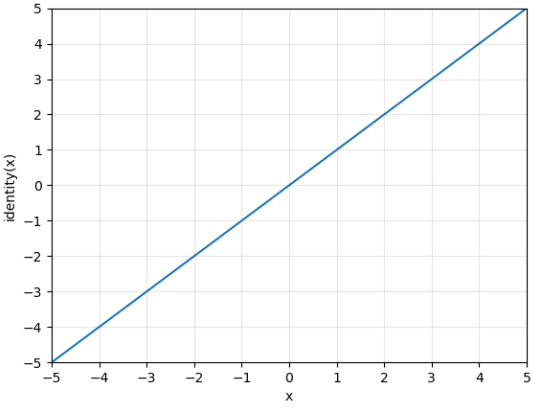
(a) Gaussian
$f(x) = e^{-(5x^2)}$



(b) Absolute value
$f(x) = |x|$

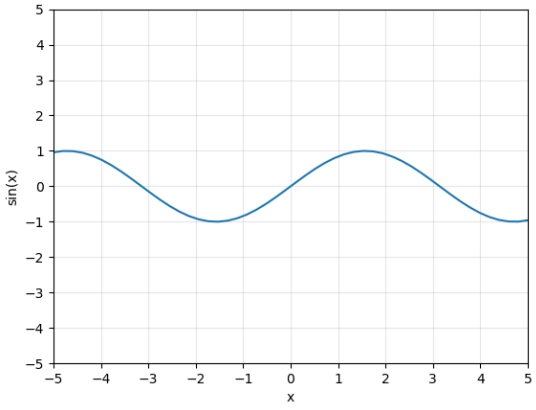


(c) Hyperbolic tangent
$f(x) = tanh(2.5x)$

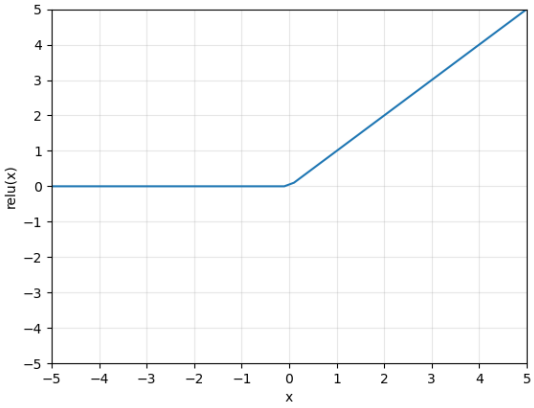


(d) Identity
$f(x) = x$

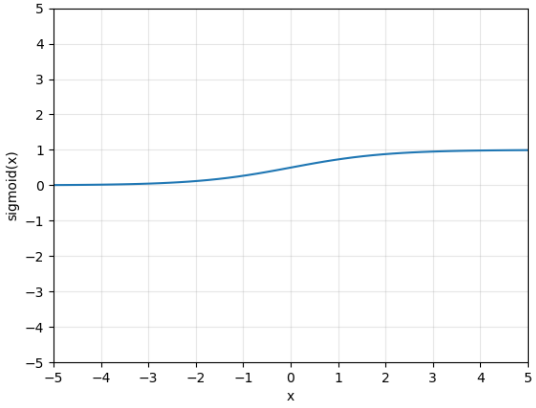
Figure A.1: A graphical overview of all activation functions made available to the evolved CPPNs presented in this work (part 1). Note that some of these functions are scaled differently compared to the canonical versions.



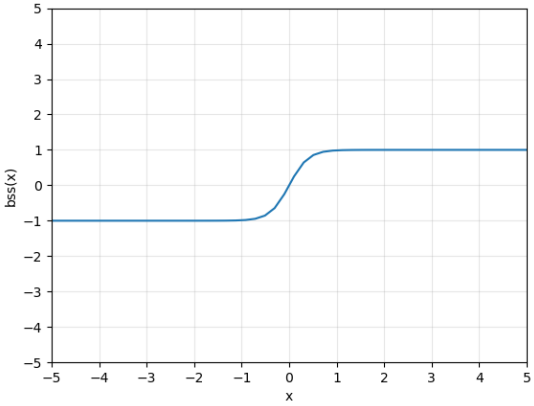
(e) Sine
$f(x) = sin(x)$



(f) Rectified Linear Unit
$f(x) = max(0, x)$



(g) Sigmoid
$f(x) = \frac{1}{1+e^{-x}}$



(h) Bipolar Steepened Sigmoid
$f(x) = \frac{2}{1+e^{-(5x)}} - 1$

Figure A.1: A graphical overview of all activation functions made available to the evolved CPPNs presented in this work (part 2). Note that some of these functions are scaled differently compared to the canonical versions.