Complex System Engineering Department (DISC)

Virtual Robots / Artificial Neural Networks / Gene Structure

Internship Report

Authors :
Matteo VACHER

Supervisors :
Pr. Claus ARANHA
Pr. Dennis WILSON

Version 1
August 7, 2026

Acknowledgements

I would like to express my sincere gratitude to everyone who contributed to making this internship a rewarding and formative experience. Above all, I am deeply grateful to my supervisors, Prof. Claus Aranha and Prof. Dennis Wilson for their continuous guidance, patience and valuable insights throughout this project. Their expertise and constructive feedback have been the corner-stone of both this work and my understanding of the field. I would also like to thank the members of the Evolutionary Computation Laboratory for their warm welcome and for shaping a stimulating research environment. The discussions I had with them greatly enhanced my approach to the problem addressed in this report. My thanks also go to the Region Occitanie that granted me a financial support with the Mouv'Occitanie mobility scholarship which made this international internship possible. Finally, I am grateful to my family and friends for their unconditional support and encouragement throughout my studies.

Contents

List of Figures

List of Acronyms

ANN	*Artificial Neural Network*
CPPN	*Compositional Pattern Producing Network*
DGA	Diploid Genetic Algorithm
GA	*Genetic Algorithm*
HyperNEAT	*Hypercube-based NeuroEvolution of Augmenting Topologies*
NEAT	*NeuroEvolution of Augmenting Topologies*
VSR	*Voxel-based Soft Robot*

Abstract

The fundamental characteristics of a Soft-Robot are its body and its controller. Both play a critical role in the final performance of the robot. Yet, parts of evolutionary approaches have chosen to consider them as distinct entities, evolving one while the other is fixed or encoding the two independently in the genome.

This report introduces a single genome, indirectly encoded representation in which one Compositional Pattern Producing Network (CPPN) generates two distinct Artificial Neural Networks (ANN) with different purposes; One builds the body, while the other controls it.

We couple this representation with a non-stationary setting, where the environment alternates between two tasks at a fixed frequency. Such settings are where diploidy and phenotypic plasticity are known to be beneficial, and they raise the question of how a single genome can reach high fitness across several environments.

Diverging from previous haploid representations of the genome, we convert this genome in a diploid form : every gene which expresses a single node in the CPPN is present with two different versions called alleles carried on two homologous chromosome as it is in a lot of cases in nature. Recombination between individuals therefore becomes meaningful, and variation can be preserved across generations without being expressed. Ploidy, however is only one of the two dimensions along which our representation varies. The second one is phenotypic plasticity, the ability of a single genotype to develop into distinct phenotype depending on the environment, which we implement in two ways : either by giving an environmental signal to the body ANN, leaving the genome untouched and read the same way in both environment, or by dedicating each chromosome to one environment so that a single chromosome can specialize on one environment. These two features are combined and studied along each other, from the case where plasticity requires no additional genetic material to the case where ploidy is itself the mechanism that produces it.

Following the HyperNEAT principle that the substrate should reflect the structure of the problem, we query the CPPN over a substrate laid along lines or circles in 5 dimensions to match the components they encode.

Lastly, we implement this approach in EvoGym, a large scale benchmark for 2D Voxel-Soft-Robot.

Introduction

The evolution of both a robot's body and controller is one of the critical challenges in Artificial Life. A first and common approach treats the morphology and the controller as two distinct entities, evolved through separate representations [1] [2]. A second line of work fixes the body and evolves only the controller [3] or the other way around, while a third approach, the closest to ours, regroups both morphology and controller within a single genome; Tanaka and Aranha [4] use the HyperNEAT algorithm to generate from one genome two separate ANN, one builds the body and the other controls it. However, Co-optimizing body and control is difficult; the morphology tends to converge prematurely while the controller is slower to converge, leading the search to then focus on a small set of bodies and morphological diversity collapses [5].

Our line of work follows the single genome implementation of Tanaka and Aranha, but diverges from it on several points motivated by the will to enrich the genome representation with nature inspired ingredients; diploidy, changing environments, phenotypic plasticity; while consciously keeping the evolutionary algorithm simple.

The first and main starting point concerns the genome representation itself. Across the VSR literature, and to the best of our knowledge, genomes have always been haploid. Diploid neural networks are themselves very rare : only three works address them [6] [7] [8], and a single one includes crossover as a step in the evolutionary process. Recombination is by far one of the central mechanisms of natural evolution, and we decided to make it the cornerstone operator of the Genetic Algorithm; by combining two parents, crossover can mix sub-structures that already work and assemble them in a new genome. To give crossover every chance to express itself, we adopt a diploid representation of the genome in which every gene exists as two alleles on two homologous chromosomes. Diploidy and recombination are in nature two sides of the same coin. The genome therefore mimics biology three times, a single genome encoding the whole organism, a diploid form associated with recombination, and phenotypic plasticity, introduced below.

Beyond recombination, diploidy is also expected to help maintain genetic diversity; recessive alleles can be preserved across generations without being expressed, providing a back-up variation that may be beneficial later. This property is reported to be particularly valuable for changing environments, where previously sub-optimal alleles can regain

advantage as conditions change. In this sense diploidy has been described as a form of long range memory [6] that keeps previous working solutions instead of rediscovering them. These benefits however tend to appear only for changing environments, which is why we introduce in our experiments a frequency at which the environment switches from one to another. Additionally, for an equal evaluation budget a diploid genome carries twice the number of alleles of a haploid representation, therefore widening the portion of the search space examined over generations.

As well as diploidy and changing environments, we also implement phenotypic plasticity, the ability of a single genotype to produce distinct phenotypes in response to an environmental variation [9]. We study it in two forms, which differ in the level at which the environmental cue reaches the individual. In the first, it enters at the developmental level : the same genetic material is expressed in both environments, and the cue, given as an input to the body ANN, interacts with the process that creates the phenotype. In the second, it acts at the level of genetic expression : each chromosome is dedicated to one environment, the input of the body ANN is a fixed bias, and the cue decides which genetic material is expressed. The added information is therefore not spread across additional genetic material in the first case, and spread across it in the second. Note that plasticity is available to the haploid individual only in the first case : in the second, it carries a single version of the CPPN and can only produce a single body and a single controller for both environments.

The simulation is based on EvoGym[2], a large-scale benchmark for 2D Voxel-Soft-Robot. The aim is that for a given environment or task, the agent's body is built by assembling different different types of blocks, called voxels, with various purposes, and controlled by expanding or reducing the size of these blocks.

In order to encode indirectly both the body and the controller, the genome of a given individual represents a Compositional Pattern Producing Network (CPPN) [10]. This study evolves the weights, biases, and activation functions of this network. The CPPN generates two Artificial Neural Networks: one that builds the body and one that directly controls the robot. Following the HyperNEAT principle that the substrate should reflect the structure of the problem [11], we query the CPPN over a substrate laid out along lines and circles in five dimensions, to match the structure of the components they encode.

Our results show that this evolutionary approach successfully co-optimizes body and controller. The main one is that a single genome can encode not only one body and one controller, but two bodies and two controllers. The second goes against our initial expectation : under developmental plasticity, the diploid algorithm does not perform as well as the haploid one. We suggest that this comes from both the body and the controller being indirectly encoded, indirect encodings being known to suffer from low locality [12] [1], on top of which diploidy adds a further source of noise. Under chromosome-level plasticity, however, the diploid representation is the one that adapts to both environments, while the

haploid one specializes on a single task instead of the two.

Because the purpose of this study is to look more deeply into the representation of the genome rather than raw performance, we deliberately make use of a simple GA with a tournament selection and elitism to evolve the population, and not a state of the art algorithm like NEAT [13]. Together, these components form a coherent pipeline in which the genome representation, rather than the evolutionary algorithm, is the object of study.

Chapter 1

Related Work

1.1 Evolution Gym

EvoGym [2] is a large benchmark for co-optimizing the design and the control of soft robots. This benchmark allows the experimenter to evaluate a robot with a given morphology and a given controller in a given environment. In EvoGym, each robot is represented by a matrix that directly encodes its body, over a two-dimensional grid. A soft robot is composed of five different types of voxels, a voxel being here one cell of that grid. The first type of voxel is empty, and represents an absence of material. Then come two types of passive voxels : a rigid one, whose shape stays close to a square, and a soft one, which can expand and contract, but only under the forces applied by its neighbors. Finally, there exist two types of actuator voxels that are driven by the controller, a horizontal one and a vertical one, whose target length is set by a value in the range $[0.6, 1.6]$, that is from 60% to 160% of their rest length. These five types are respectively encoded by a value in $\{0, 1, 2, 3, 4\}$.

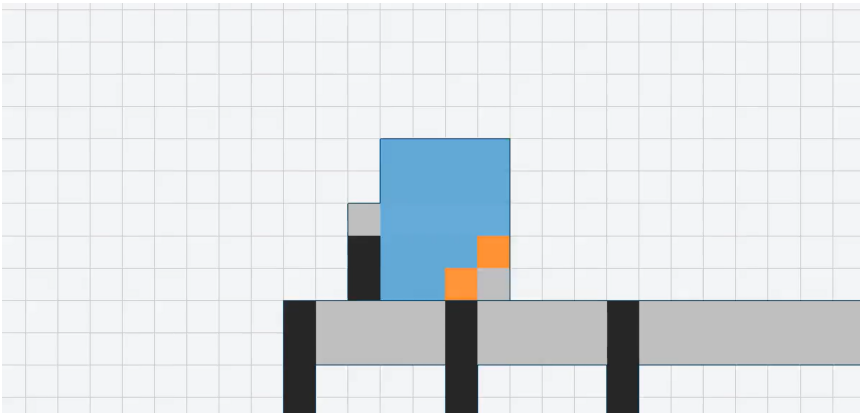


Figure 1.1: Example of a soft robot in the BridgeWalker-v0 environment of EvoGym.

The robot of Figure 1.1 is represented by the following matrix : $\begin{bmatrix} 0 & 4 & 4 & 4 & 4 \\ 0 & 4 & 4 & 4 & 4 \\ 2 & 4 & 4 & 4 & 4 \\ 1 & 4 & 4 & 4 & 3 \\ 1 & 4 & 4 & 3 & 1 \end{bmatrix}$

At each time step, the robot acts according to the output of the controller, which is computed from the observation vector returned by EvoGym. This vector contains several kinds of information, some of which depend on the environment the robot is placed in.

- **Position** : the observation vector contains the positions of the point masses located at the corners of the voxels composing the robot.
- **Velocity** : it also contains the velocity of the center of mass.
- **Orientation** : another descriptor that can be added to the observation vector is the orientation of the robot relative to its initial orientation.
- **Terrain information** : in tasks that require the position of obstacles, the observation vector also describes the elevation of the terrain around the robot.

The availability of these sensors depends on the task at hand, but the positions of the point masses are a consistent input across all the environments. Tasks in EvoGym are ranked by difficulty : walking on flat terrain is labelled as easy, while climbing stairs is labelled as hard. Nevertheless, some tasks ranked as medium can be harder to solve than tasks ranked as hard. The method used to solve the problem plays a major role here, as some methods perform better on hard-ranked tasks while yielding poor results on medium-ranked ones.

EvoGym features various types of tasks, listed below :

- **Walking**, these tasks require the robot to walk forwards and backwards on relatively flat terrain. Those tasks are usually easy.
- **Object Manipulation**, in this type of environment, the robot is interacting with another object to achieve its goal. Usually it involves carrying, pushing, throwing, catching, lifting and more, and these environments are often referred to as hard.
- **Climbing**, in these tasks, the robot must climb upward on varying types of terrain that are usually ranked as medium.
- **Locomotion**, these environments force the robot to traverse different types of terrain including steps going up and down, bumpy terrain, terrain with obstacles, terrain with gaps and holes and more. These environments usually vary in difficulty between medium and hard.

- **Shape Change**, these tasks require the robot to minimize or maximize its shape. These tasks are usually easy.
- **Miscellaneous**, these environments usually require the robot to flip, jump or balance and are easy.

Several works have used EvoGym to study different aspects of the evolution of soft robots [4] [14] [1].

1.2 Direct and Indirect Encoding

Encoding in evolutionary computation refers to how the genotype maps to the phenotype. Direct encoding is, at first sight, the simplest approach : each gene corresponds to one parameter of the phenotype. For example, if the controller of a robot is a neural network with one hundred weights, the genome will contain one hundred genes, one per weight, as shown on Figure 1.2. It is easy to implement but it scales poorly. As the neural network gets more complex, so does the genome, and there is no structure or regularity that can be exploited [15]. An example of direct encoding for the body of a soft robot can be found in [1].

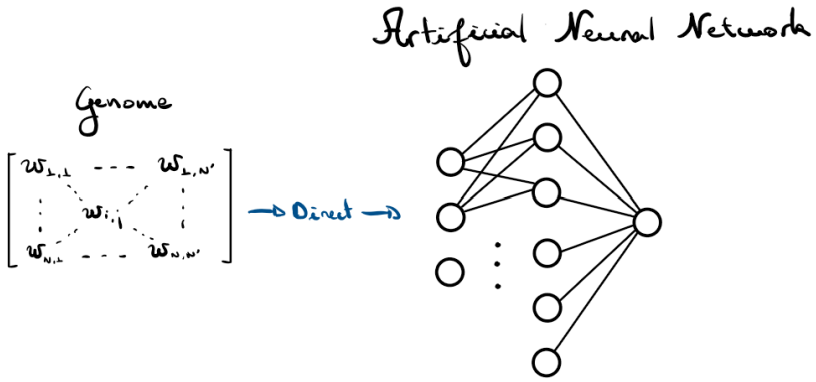


Figure 1.2: Example of Direct Encoding

Indirect encoding, on the other hand, adds a level of abstraction which gives the possibility to compress the genome as in nature. For example, the human body is composed of 20 000 genes [16] but its neocortex alone has more than 1.5×10^{14} connections [17]. The genome does not describe the phenotype directly, but instead encodes a set of rules or a function whose job will be to generate this phenotype. The key advantage here is that it is compact and has the capability of capturing the regularities of the problem. A typical example lies in the generation of the body in the EvoGym benchmark. Many works [2] [1] [4] have chosen to use an indirect representation of the phenotype. For instance,

the genome could be a direct representation of a neural network, and this neural network would act as an intermediary by designing the body of the robot, as shown on Figure 1.3.

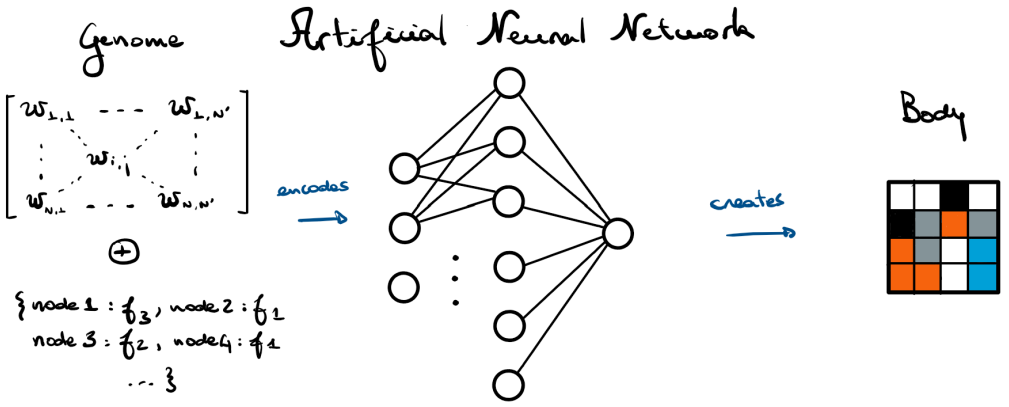


Figure 1.3: Example of Indirect Encoding

This is the family of encoding we use in this work, and this is also what makes the crossover interesting : recombining two genomes means recombining two ways of building a body, rather than two lists of parameters. Indirect encodings are known to suffer from low locality [12], so a small change in the genotype can lead to a large change in the phenotype. We observe nonetheless that children keep a strong resemblance to their parents, which suggests that the crossover does transmit something of what already works. We come back to this point in the results section.

1.3 HyperNEAT

A Compositional Pattern Producing Network (CPPN) is a neural network that maps geometric coordinates to output values, and whose activation functions are chosen from a rich set including, for example, sine, Gaussian and tanh, allowing it to reproduce spatially regular and symmetric patterns [10]. Rather than encoding a phenotype directly, a CPPN encodes the rules that generate it : it is queried over a spatial substrate and produces a structured output that mimics the regularities found in biological morphologies.

NEAT stands for NeuroEvolution of Augmenting Topologies and is an algorithm that evolves both the topology and the components of a neural network [13]. NEAT is based on three key ideas :

- **First**, in order to increase the complexity of the neural network structure, a method is implemented to keep track of the changes occurring in the neural network topology. It does so in order to allow individuals to make crossover with each other. Let us

remind the reader at this point that NEAT performs both crossover and cloning-mutating as ways to obtain a new individual. In order to perform crossover, NEAT assigns a unique historical marking to every new piece of neural network structure that appears through a mutation. This solves the problem of matching different topologies in an evolving population.

- **Second**, NEAT speciates the population so that individuals that have a similar genome can compete with each other, therefore allowing each individual in its own niche to be better than its closest neighbors instead of interfering with the evolution of completely distinct individuals. This way, the algorithm protects new innovations that may not be relevant at the moment but may prove their utility in the future.

- **Third**, NEAT begins with a uniform and simple population composed of individuals with no hidden layer, differing only in their initial weights. Over the course of evolution, different topologies are created, leading to diverse and complex phenotypes.

CPPN-NEAT is therefore an extension of NEAT that allows it to evolve CPPNs [10]. These algorithms are almost the same but differ in that, in the second version, an activation function is assigned to each node, whereas before sigmoid was the only possibility. Another clear difference is that the activation functions of each node now take part in the distance between two individuals, used for the speciation of the population.

HyperNEAT is an indirect encoding method proposed by Stanley et al. [11] that evolves a CPPN whose purpose is to generate the weights of a neural network from the geometric position of its nodes on a substrate, as shown on Figure 1.4. The main idea here is that the structure of the controller should reflect the geometry of the problem. By placing the neurons in a space whose layout represents the geometry of the problem at hand, the CPPN can exploit the regularities of the task.

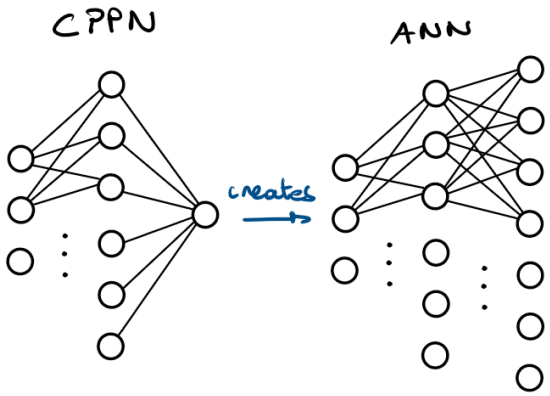


Figure 1.4: Process of creation of the artificial neural network.

In the context of VSRs, HyperNEAT is particularly well suited since the body of the robot is naturally laid out on a two-dimensional grid. The substrate can therefore directly reproduce this pattern, an example of which is given on Figure 1.5. Our approach takes inspiration from this algorithm but deliberately diverges from it : we use a fixed-size CPPN and a simple genetic algorithm, and we do not evolve the topology of the CPPN. Its structure stays the same across the whole run, and what evolves are its weights, biases and activation functions.

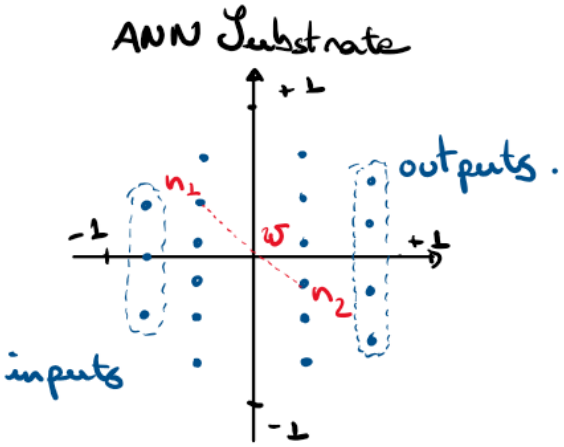


Figure 1.5: An example of a representation of an ANN in a two-dimensional substrate. Here, the two coordinates of nodes n_1 and n_2 are concatenated to form the CPPN input. This CPPN will return the weight w between these two nodes. Please note that here, an ANN with two hidden layers, a three-dimensional input and a four-dimensional output is represented through the substrate.

1.4 Single Genome in EvoGym

Tanaka and Aranha [4] proposed a new method to encode both the body and the brain of the robot in a single genome, compressing the genome of an individual even further. To perform such an encoding, they used HyperNEAT to evolve a CPPN (both its topology, weights, biases and activation functions) that produces two artificial neural networks. One generates the body and the other controls it. In order to create both ANNs from a unique substrate, they use a three-dimensional substrate, whose last coordinate indicates both the current layer and the current ANN it encodes. The morphology ANN was derived from a positive last coordinate, while the controller ANN was derived from a negative last coordinate, as shown on Figure 1.6.

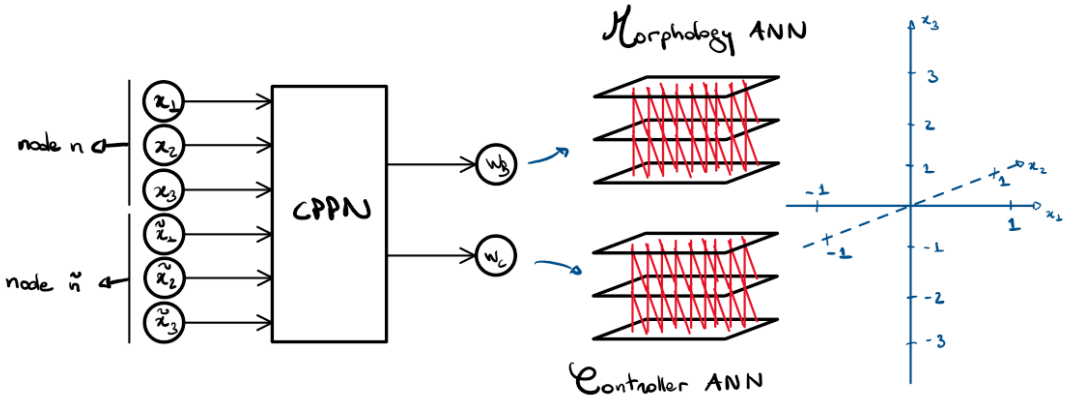


Figure 1.6: Representation of the genome and how it encodes two ANNs.

Thanks to the body generated by the morphology ANN and to the controller, both provided by the CPPN, EvoGym evaluates the robot in a given environment and gives it a fitness. NEAT is then applied to the CPPN to create the next generation of individuals. This configuration has been tested on four different environments :

- **Walker-v0**, the objective for the robot is to go as far as possible. Results showed that many different morphologies were found by the algorithm, but the best individuals would always behave the same way. Mostly composed of vertical actuators and performing big jumps, they suggest that the algorithm has found an easy and yet really good configuration.

- **ObstacleTraverser-v1**, an environment similar to the Walker-v0 task, but which differs due to the presence of an irregular ground that makes it difficult for the robot to cross the whole distance. However, the same type of highly performing robots was discovered.

- **Climber-v2**, an environment where the robot has to climb as high as possible with the help of two parallel walls. Unfortunately, robots were unable to evolve correctly, and again, the best performing robots were made of vertical actuators and were moving by jumping. This is consistent with the findings of Pigozzi et al. [1], who noted that evolution tends to favor morphologies composed predominantly of actuators, with the fraction of active voxels increasing over the course of evolution.

- **Thrower-v0**, a task where the robot has to throw a block of solid voxels as far as possible. In this task, a bigger variety of high performing robots was observed, leading for example to a robot with two towers, one composed of solid voxels and the other of vertical actuators.

1.5 Haploidy and Diploidy

Phenotypic plasticity is the ability of a genotype to produce different phenotypes depending on the environment it is placed in [9]. It is found in all domains of life, and it can change the shape of an organism as well as the way it behaves. Sommer explains that this variation can be continuous, or discrete when the organism has only a fixed number of possible forms, one per environment. In that second case, the response usually comes from a switch gene, whose job is to sense the environment first, and then to decide which form is expressed.

Ploidy refers to the number of sets of homologous chromosomes per cell : haploid organisms carry one set, while diploid organisms carry two [18]. These two ploidy levels show different evolutionary properties along two key axes : the number of mutations and the efficiency of selection.

- **Mutations**, diploid individuals present twice the number of mutations compared to haploids, generating more mutations overall. Mutations in a diploid population may occur but not be expressed in the phenotype, if the mutation itself belongs to a recessive allele, therefore giving a form of latent diversity [18].
- **Selection efficiency**, for haploid individuals, every mutation is directly visible in the phenotype, thus allowing it to be tested in the environment and making it visible for selection. In a diploid population, a deleterious mutation can be masked by a dominant allele, slowing the process of removing deleterious alleles from the population [18].

However, this masking effect, illustrated on Figure 1.7, is precisely what gives diploidy a key advantage when it comes to changing environments. Each recessive allele serves as a backup solution if a change occurs in the environment.

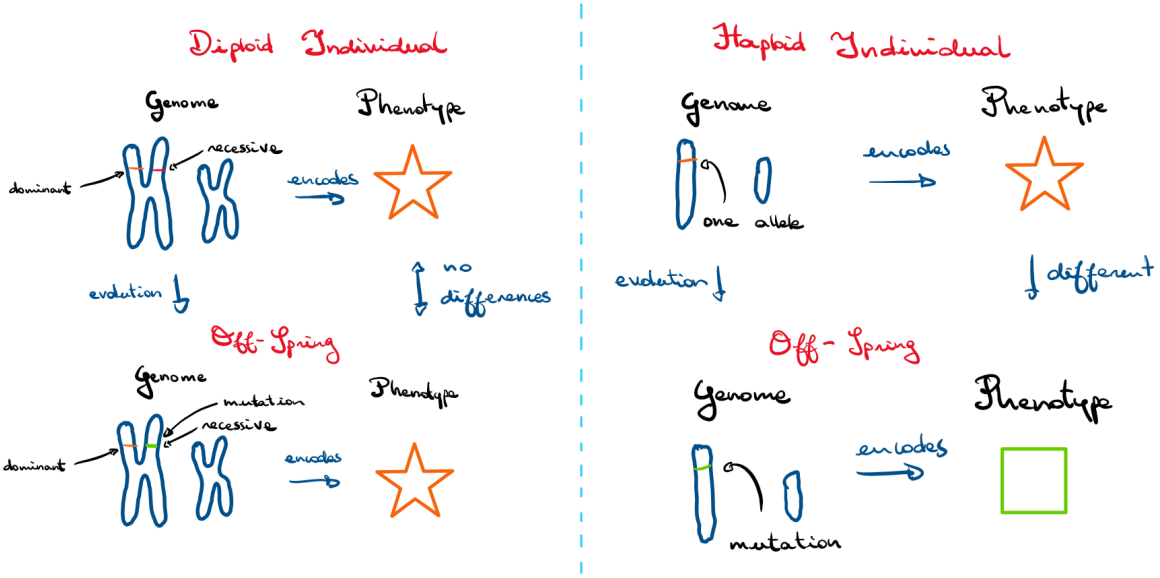


Figure 1.7: Example of the consequences on the phenotype when a mutation occurs in haploid or diploid genotypes.

1.6 Diploidy in Neural Networks

Before diving into the related work on diploid neural networks, a short detour regarding diploid genetic algorithms is necessary.

Diploid Genetic Algorithms (DGAs) represent a biologically inspired extension of usual genetic algorithms, where the genome contains each piece of information twice, one copy being usually expressed via dominance mechanisms, with the objective of enhancing adaptability, diversity and long-term memory [19]. The reason for this long-term memory is that when a change in the environment occurs, unexpressed alleles can be reactivated and exploit a solution that worked before. In a very large survey including over 160 studies, Matei et al. [19] confirmed that DGAs consistently outperform haploid GAs in dynamic optimization, while their advantage in static settings remains unclear.

Among the many domains surveyed, robotics and neuroevolution are the most relevant to our work. DGAs have been applied to evolving robotic controllers, providing a mechanism for maintaining diversity and reactivating old successful strategies when the environment changes. Despite these promising results, the integration of diploid representations with neuroevolution frameworks such as NEAT or HyperNEAT remains largely unexplored [19]. Moreover, at this time there exist only three papers studying the impact of diploidy and haploidy on neural networks.

The first work applying diploidy specifically to neural networks is that of Calabretta et al. [6], which remains a key reference and lays the basics of this discipline. Haploid and diploid populations are compared on a simulated robot tasked with exploring an environment and periodically returning to a food source, in both fixed and changing conditions. Each chromosome pair contains an encoding of the weight values and two dominance modifier genes resolving conflicts between the two homologous chromosomes.

- In fixed environments, haploids achieve better average results, but diploids reach higher peak fitness, both for a low mutation rate of the dominance.
- In changing environments, diploid populations adapt more robustly, as unexpressed genes preserve a genetic memory of solutions that worked for other environments, which is consistent with the survey of Matei et al. [19]. Diploids at a 1% mutation rate outperform haploids at 3% on both average and peak fitness, but when compared to haploids at 1%, they only win on peak fitness : no single haploid population can achieve both simultaneously.

Following the path laid by this research, Calabretta et al. [7] drastically increased the frequency of the environment changes. Diploid populations exhibited very little fitness variation compared to haploids, and appeared to find a solution suited for both environments simultaneously rather than alternating between them.

Reedy directly extends the work of Calabretta et al. by introducing a recombination between diploid genomes [8], and testing two distinct diploid variants against a haploid baseline. The first variant initializes each agent with two identical copies of the same chromosome, taken from a pre-trained set of evolved individuals, while the second assigns two different chromosomes from the same set. In the implementation, all genes share the same percentage of mutations, but a conditional probability indicates which mutation shall occur. For instance, if a mutation has to occur on a weight gene, there is more chance that this mutation will affect the value of the weight itself rather than the value of the dominance associated with this allele. The three genetic systems are evaluated across three different environments : one that is static, another one that is changing, and the last one, called nutrition environment, that displays two alternating food sources of different quality. Results are counter-intuitive : diverse diploidy outperforms haploidy in all three environments, in contradiction with the previous results stating that haploidy would perform better in static environments. On the other hand, plain diploidy performs best in the nutrition environment. Haploidy never achieves the best performance in any condition, and lies in between the two diploid forms.

Chapter 2

Method

The main objective of this study is to investigate whether a diploid genome representation, combined with phenotypic plasticity, offers a measurable advantage over a haploid one in changing environments, in the context of an indirect encoding where a single CPPN generates both the morphology and the controller of a soft robot.

Rather than seeking a definitive answer, this work aims to identify whether and under which conditions diploidy is beneficial, and to better understand how an indirect encoding, where small changes in the genotype can lead to large changes in the phenotype, makes the effect of diploidy difficult to isolate from the other factors involved in the evolution.

This question is framed within a specific representational constraint. Tanaka and Aranha [4] encode two entities, a body and a controller, within a single genome. We retain this single-genome formulation but extend it to a changing environment, so that the same genome must now encode two bodies and, as a consequence, two controllers.

First, the added information is not spread across additional genetic material : even in the diploid condition, no chromosome is tied to a given environment. It is absorbed by the same CPPN, which is given no explicit mechanism to separate what belongs to one environment from what belongs to the other. The only element that carries the environmental change is a single cue, given as an input to the body ANN : the genome is read in the same way in both environments. How far this formulation can be extended before it saturates is the question this work sets out to address experimentally.

Second, the added information is spread across the additional genetic material : the diploid genome stores on each chromosome a complete CPPN that is expressed depending on the environment the population is placed in. The first chromosome is only used for the first environment while the second is only used for the second task. The input to the body ANN is for this process a fixed bias that never changes. Therefore the haploid version of this algorithm is only able to create a single body and a single controller per individual.

Both methods are a form of phenotypic plasticity, defined as the ability of a single geno-

type to produce distinct phenotypes in response to an environment variation [9]. What separates the two mechanisms is the level at which the environmental signal is sent to the individual. In the first, it enters at the developmental level : the same genetic material is expressed and the environmental signal interacts with the process of creation of the phenotype. In the second, it acts at the level of the genetic expression, and the environmental signal decides which genetic material is expressed. Note also that this plasticity is available for the haploid individual only in the first case, in the second, the haploid genome carries a single version of the CPPN and therefore, because of a single cue as the input of the body ANN, can only produce a single body and a single controller for two environments.

2.1 Developmental Plasticity - An Overview

2.1.1 The Main Inspiration

The method presented here builds on the work of Tanaka and Aranha [4], who use HyperNEAT to evolve CPPNs that generate both the body ANN, which creates the body, and the controller ANN, which pilots it. This method successfully evolved CPPNs on several tasks. Tanaka observed that many of the successful bodies generated by the CPPNs contained a large proportion of vertical actuators. Such morphologies appear to be an easy solution for both parts of the problem : they are easily produced by the CPPN, and they only require a simple jumping controller to perform well across several environments.

The second main source of inspiration is the work of Calabretta et al. [6] and Reedy [8], which lays the foundations of diploid neural networks. Building on these approaches, we evolve a diploid CPPN.

2.1.2 Developmental Plasticity - The Overall Method

Motivated by the study of changing environments, we implement a diploid CPPN. We keep its topology fixed, so that the effect of diploidy can be observed without the structural mutations of NEAT interfering with it. This fixed topology allows us to represent a whole population of diploid individuals with a uniform genome structure, and results in a simpler variant of the HyperNEAT algorithm, in which the topology is not evolved. From a five-dimensional substrate, this CPPN designs two ANNs. The first one creates the VSR body, while the second one is the controller that pilots it.

As a baseline for comparison, we also implement a haploid version of the CPPN that produces the exact same phenotype through the same process and differs only in the ploidy of its genome.

Since our focus is on the representation of the genome rather than on the method used

to evolve individuals, we deliberately keep the evolutionary algorithm simple and pair this genome with a standard genetic algorithm.

2.2 Developmental Plasticity - The CPPN

2.2.1 Architecture of the CPPN

Our CPPN is a simple ANN composed of a first hidden layer fully connected to the inputs, the second layer also fully connected is followed by two branches : one leads to the output that returns the weights of the body ANN, while the other leads to the output responsible for the controller. The reason is the following : a fully connected first layer allows each node to form its own interpretation of the coordinates given to the CPPN, while splitting this trunk into two separate branches prevents the two outputs from interfering too much with each other. Preliminary experiments suggest that this design yields better overall results. The CPPN takes eleven inputs, formed by the concatenation of three vectors : the coordinates of two points in five dimensions, and the distance between them. The first layer contains five nodes, the second consists of two branches of three nodes each, and the last layer contains two nodes, one on each branch, as shown on Figure 2.1.

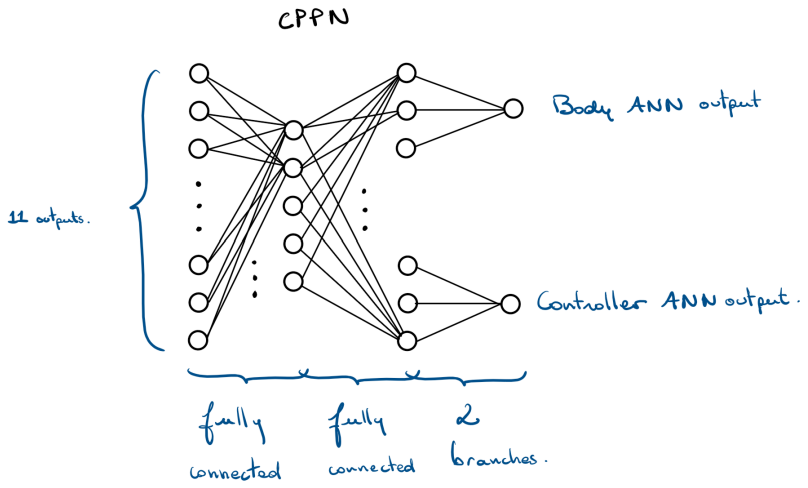


Figure 2.1: Architecture of the CPPN evolved in this work

2.2.2 Developmental Plasticity - The Substrate

Following the previous method, the CPPN encodes two ANNs within a single substrate : one can be seen as the above-ground part of a building, while the other represents the underground part. The network to which a node belongs is determined by the sign of its

first coordinate. Here, we further restrict the range of this coordinate: a node of the body ANN has a first coordinate in $[1/3, 1]$, while a node of the controller has a first coordinate in $[-1, -1/3]$. Keeping this coordinate away from zero ensures that the CPPN always receives a non-negligible input on this dimension, whichever network is being queried.

Tanaka and Aranha note that some environments favour irregular bodies, which their method could not produce. They also suggest that future work should look for new ways to build a body from a network. To increase the irregularity of the morphology, and therefore to evolve more sophisticated bodies, we propose a five-dimensional substrate whose purpose is to allow more complex bodies to be produced by the ANN. Our supposition is that, by taking the coordinates of the encoded voxel as inputs, the body ANN of Tanaka and Aranha [4] is left to organise the geometry of the body by itself, rather than this geometry being encoded directly by the CPPN. The CPPN then only reaches the shape of the body through the weights of a network that computes it, so that the body is indirectly encoded twice rather than once. In our representation, the input of the body ANN carries no spatial information : it is a single value, which varies with the environment and takes the value $\sqrt{2}/2$ or $-\sqrt{2}/2$. The position of each voxel is instead expressed in the substrate itself, so that the geometry is encoded directly by the CPPN. This makes the generated body more deterministic from the point of view of the CPPN, and allows it to design two bodies for two environments, which we hypothesise allows better adaptation.

A second limitation concerns the regularity of the body. The former representation tended to produce bodies with a high proportion of vertical actuators, if not exclusively so. A possible explanation lies in the way each voxel type is represented. In the work of Tanaka and Aranha [4], the voxel type at a given position is determined by the position of the maximum value among the outputs of the body ANN, which leaves five choices, but these five outputs are laid out as a line in the substrate. Moreover, the two outermost outputs are the ones representing the empty voxel and the vertical actuator. Based on the observations made in our early experiments on the same substrate, we suggest that this way of representing the outputs favours the points located at the extremities of the substrate. In order to give every voxel type the same chance, we represent the five outputs on a circle. This places all voxel types on an equal footing and leaves the body ANN free to produce more complex bodies.

Concretely, each neuron is described by a five-dimensional coordinate $(d_1, d_2, d_3, d_4, d_5)$. The first component d_1 encodes the layer (by the magnitude), its sign separating the body ANN ($d_1 \in [1/3, 1]$) from the controller ANN ($d_1 \in [-1, -1/3]$), the pair (d_2, d_3) maps the neuron onto the x–y grid of the robot, normalised in $[-1, 1]$, and the last pair (d_4, d_5) places the five voxel types on a circle of radius $\frac{1}{2}$, collapsing to $(0, 0)$ everywhere except on the output layer of the body ANN, as shown on Figure 2.2.

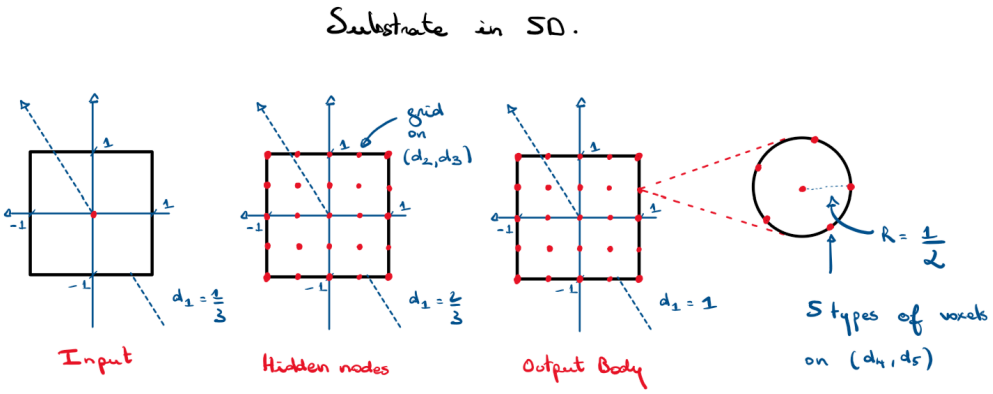


Figure 2.2: The part of the 5 dimensions substrate representing the Body ANN

The controller ANN follows the same model as Tanaka and Aranha [4], but with a hidden layer that reflects the geometry of the problem, in order to help the CPPN capture the fact that the robot is a cubic soft robot.

2.3 Developmental Plasticity - Diploid Neural Network

2.3.1 How to double every gene ?

In order to create a diploid CPPN, every bias, weight and activation function must exist in a second copy within the genome. To achieve this, we implement two versions of each node of the CPPN, as shown on Figure 2.3.

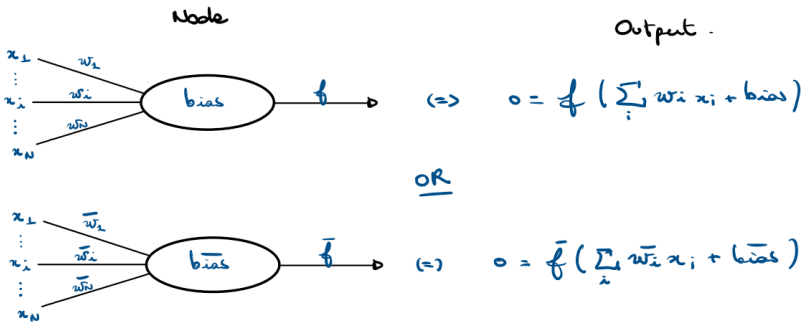


Figure 2.3: The co-existence of two versions of the same node in a single genome

Every node exists in two copies, one on each of two homologous chromosomes. Unlike a human genome with its 23 pairs, our individuals carry a single homologous pair, so each node's two versions fully describe the individual's genotype. The output of a node is

described by the following formula :

$$o = f\left(\sum_i w_i x_i + b\right) \tag{2.1}$$

where :

- f **is the activation function of the node**, three activation functions are available in this CPPN : *tanh, gauss, sin*. The activation function of a node can change according to its own mutation rate.
- x_i **are the inputs coming from the nodes of the previous layer**.
- w_i **are the weights associated to the inputs of the node**, they are initialized in $[-1, 1]$ and have their own mutation rate.
- b **is the bias associated to the node**, it is also initialized in $[-1, 1]$ and also has its own mutation rate.

In our representation, a gene is a whole node : its weights, its bias and its activation function are inherited together as a single unit.

All this information acts as a group : it lies on the same chromosome and together defines a node. In a diploid individual, every node is therefore implemented with its two sets of weights, activation functions and biases, carried on two homologous chromosomes, as shown on Figure 2.4.

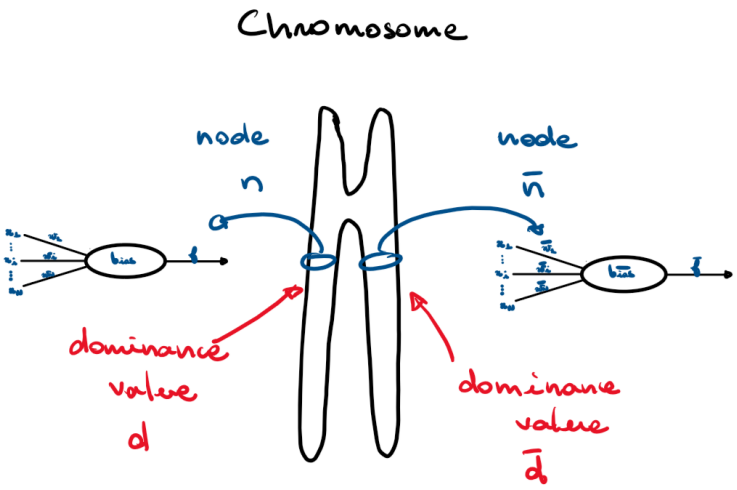


Figure 2.4: A pair of homologous chromosomes, showing two copies of the same node and their respective dominance values, which determine which allele is expressed

2.3.2 Dominance

The CPPN's genome is now composed of several nodes, each of them existing in two versions. To decide which version is expressed in the CPPN, we introduce a dominance mechanism. Each node carries an additional value between 0 and 1 that describes its dominance over its homologous node. Dominance is decided node by node : the two chromosomes are not ranked as a whole, and the expressed genome mixes nodes from both. The rule is the following :

```
expressed_genome = []
for each node :
    if chromosome1[node][dominance] > chromosome2[node][dominance] :
        expressed_genome.append(chromosome1[node])
    elif chromosome1[node][dominance] < chromosome2[node][dominance] :
        expressed_genome.append(chromosome2[node])
    else :
        expressed_genome.append(chromosome1[node])
```

When both dominance values are equal, the node of the first chromosome is expressed. This is not a bias towards one chromosome, since the two chromosomes are chosen randomly : it is a way to keep the group activity of a sub-space of the Neural Network. By doing so, if many dominance values are equal, the nodes expressed will all be on the first chromosome, to avoid breaking too many substructures.

2.3.3 Genetic Algorithm

According to Pretorius and Pillay [20] and to earlier literature, crossover operators are often omitted from genetic algorithms applied to neural networks, as they tend to be highly destructive for the network. Keeping this in mind, we nonetheless implement a fully crossover-based GA, which means that every new individual is the offspring of two parents and the result of a mutation, as shown on Figure 2.5.

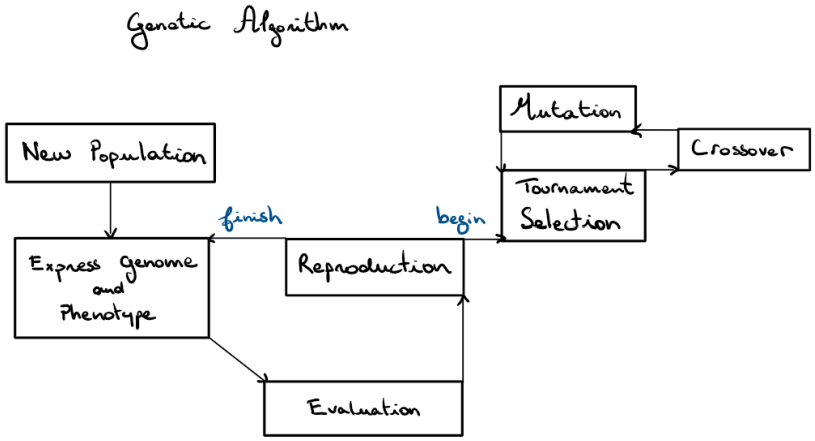


Figure 2.5: A diploid Genetic Algorithm using only a crossover operation to generate offsprings

The crossover operation used in this work is a form of reproduction based on meiosis. During meiosis, a chromosome can be cut and recombined with its homologous chromosome. In nature, the exact point at which chromosomes are cut is not fixed, yet in the process of meiosis implemented we cut each chromosome exactly between the first and the second layer. The exact process is described below and illustrated on Figure 2.6 :

- **First**, separate each homologous chromosome at a fixed point which is right after the last node of the first hidden layer and right before the first node of the second hidden layer.
- **Second**, reassemble the resulting parts crosswise : the first part of the first chromosome is linked to the second part of the second chromosome, while the first part of the second chromosome is linked to the second part of the first chromosome.
- **Third**, randomly select one of the two recombined chromosomes, for each of the two individuals involved in the mating process.
- **Fourth**, attach these two chromosomes to the offspring, which thus receives one homologous chromosome from each parent.

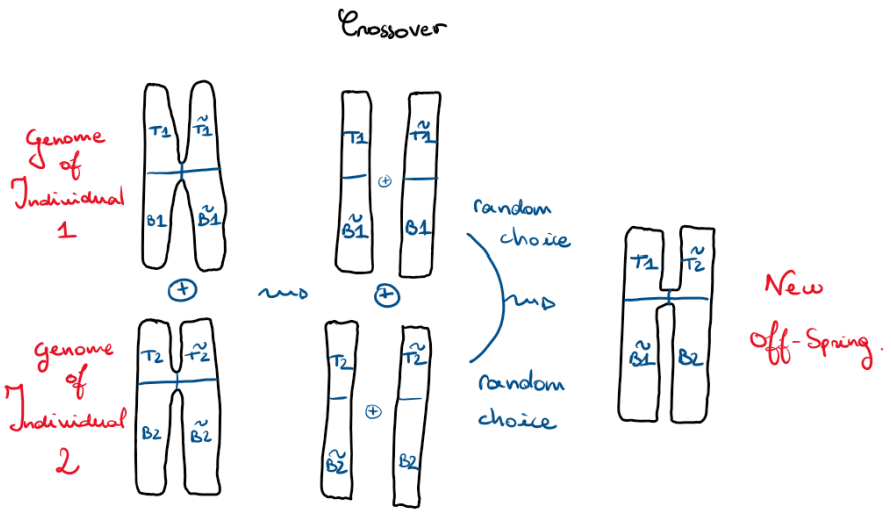


Figure 2.6: The crossover of diploid individuals

The mutations applied to the CPPN consist of a Gaussian noise with mean 0 and standard deviation σ. Every type of information in the genome has its own standard deviation, and this standard deviation does not depend on whether the information is carried by the recessive or by the dominant node. The weights, the biases, the activation functions and the dominance values therefore each have their own standard deviation.

To trigger the Gaussian noise, we draw a random variable from a uniform distribution over $[0, 1]$: if the draw is below a given threshold, the Gaussian noise is added to the current value. This mechanism is used to modify the weights and the biases. The dominance follows the same process with one difference : if the new value falls outside the allowed range $[0, 1]$, the excess is reflected back inside the range. For example, if the previous value is 0.96 and the sampled noise is 0.1, the new dominance value is 0.94 rather than 1.06.

As for the activation function of the node, a new one is selected from the set of available functions and replaces the previous one.

Each type of information also has its own threshold.

2.4 Developmental Plasticity - Haploid Neural Network

2.4.1 The Encoding

In this configuration, the genome of a haploid individual is composed of a single chromosome. All the nodes present in the genome are therefore also the ones expressed in the CPPN.

2.4.2 Genetic Algorithm

Unlike the diploid genetic algorithm described previously, the haploid version does not use a dominance value in any form, neither for the crossover mechanism nor for the expression of the genome.

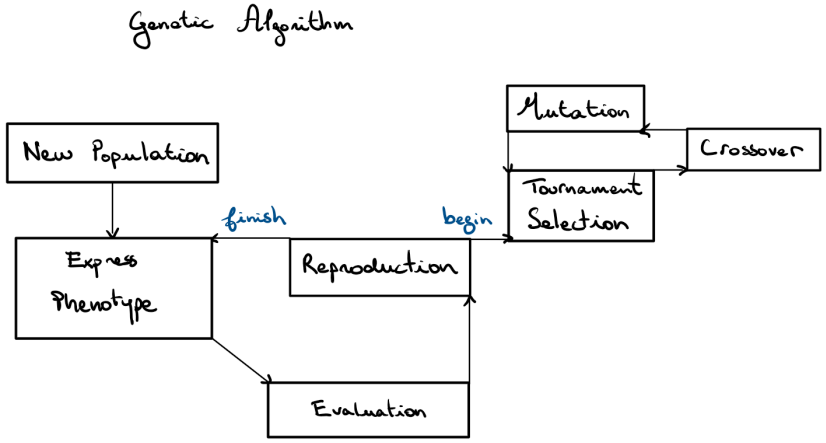


Figure 2.7: Figure representing the Haploid Genetic Algorithm

As shown on Figure 2.7, the genetic algorithm is simply : evaluate, reproduce via crossover with tournament selection, and repeat. For the haploid population, the crossover between two individuals proceeds as follows :

- **First**, take the two individuals involved in the crossover and align each node with its homologous node in the other parent.
- **Second**, for each pair of nodes, randomly select one of the two and pass it to the offspring.

Right after the crossover phase, each piece of information is mutated in the same way as in the diploid algorithm, with exactly the same thresholds and standard deviations.

2.5 Developmental Plasticity - Environment and Experiment

2.5.1 How to encode two environments in a single genome ?

This fixed topology allows us to represent a whole population of diploid individuals with a uniform genome structure, and results in a simpler variant of the HyperNEAT algorithm,

in which the topology is not evolved. We refer to this variant as Minimal Hypercube-based Neuroevolution.

Since diploidy is mainly expected to improve adaptation to changing environments, we implement this approach in EvoGym using two environments that alternate during a single evolutionary run. Based on preliminary experiments, we selected environments that are expected to lead to different types of morphology and controller, in order to study the impact of these encodings under the clearest possible conditions. The robots are therefore evaluated on the Walker-v0 and Pusher-v0 tasks. The Walker-v0 task has a well-known solution, in which the robot moves by performing large jumps ; in the Pusher-v0 task, on the other hand, a robot that jumps obtains a poor score, as it ignores the box.

When the environment changes, the input of the body ANN is switched, so that the same genome expresses a different body. The controller substrate follows this change, since it is defined by the morphology : the CPPN is then queried at coordinates that were never under selection in the other environment. The same genome must therefore account for two bodies and two controllers, which is the constraint this work sets out to examine.

Figure 2.8 describes how a single expressed genome can encode two phenotypes adapted to two different environments. This way of encoding can be referred to as Developmental Phenotypic Plasticity since the environment changes the expression of the genome and not the genome itself.

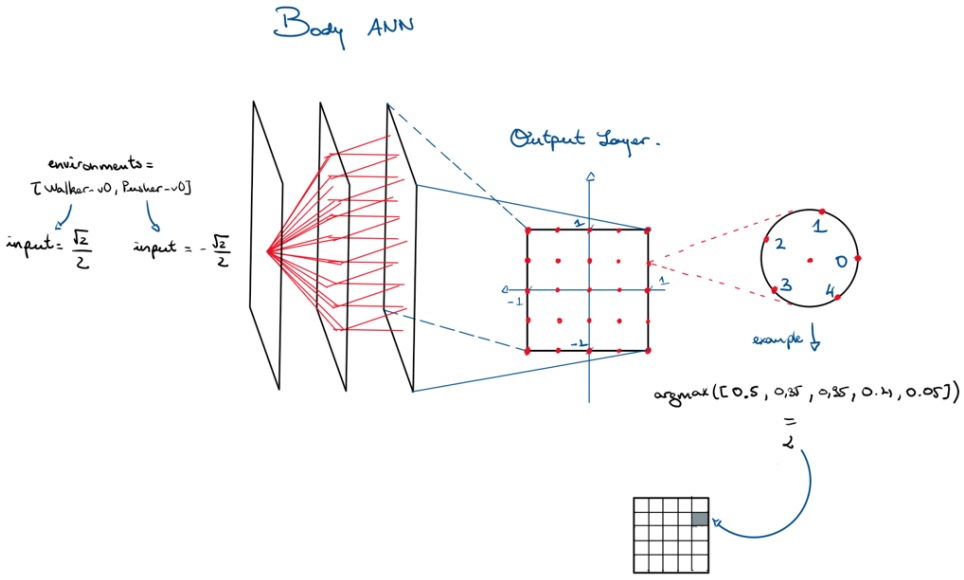


Figure 2.8: Figure representing the switch of input of the body ANN that follows the switch of environment

2.5.2 Parameters

The parameters used in the experiments are gathered in Table 2.1. They are identical for the haploid and the diploid populations, and for the two plasticity conditions, so that the only difference between two runs lies in the representation of the genome itself. Every experiment is repeated over 5 independent runs.

Parameter	**Value**
Evolution	
Population size	128
Number of generations	600
Environment switch	every 20 generations
Independent runs	5
Tournament size	2
Winners per tournament	1
Elites kept per generation	1
Reported individuals	40
Genome	
Activation function pool	gaussian, sin, tanh
Initialisation range, weights	$[-1, 1]$
Initialisation range, biases	$[-1, 1]$
Bound on the weights	$[-20, 20]$
Bound on the biases	$[-20, 20]$
Mutation	
Threshold, weights	0.03
Threshold, biases	0.01
Threshold, activation functions	0.01
Threshold, dominance	0.07
σ, weights	0.1
σ, biases	0.1
σ, dominance	0.1
Evaluation	
Episode length, Thrower-v0	300 time steps
Episode length, Pusher-v0	1000 time steps
Episode length, Walker-v0	500 time steps

Table 2.1: Parameters used in the experiments

2.6 Chromosome-level plasticity - Diploid Neural Network

As in the developmental plasticity condition, every node exists in two copies, located on the two homologous chromosomes. When the environment switches, the input of the body ANN does not change and remains fixed at $\frac{\sqrt{2}}{2}$.

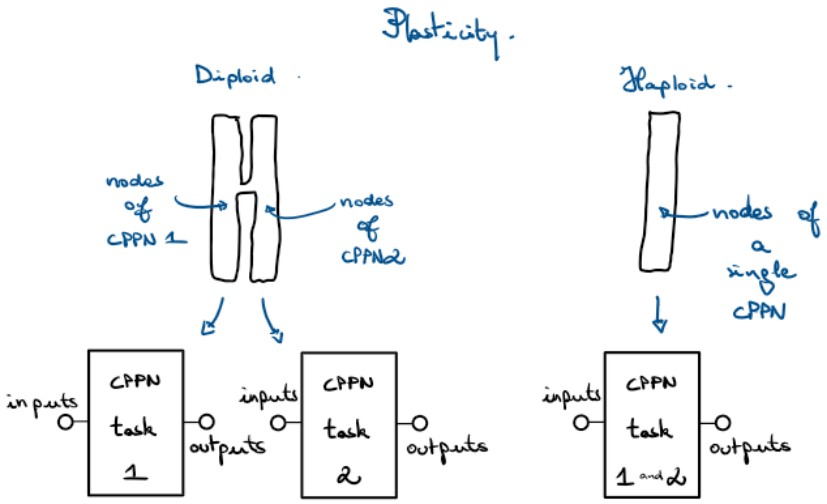


Figure 2.9: Chromosome-Level Plasticity over diploid and haploid genome

Instead, the first chromosome is always the one expressed in the first environment, while the second chromosome is always the one expressed in the second environment, as shown on Figure 2.9. To ensure that the chromosomes associated with the two environments are not mixed, the crossover of two diploid individuals recombines their first chromosomes together, and their second chromosomes together. The procedure is described below and on Figure 2.10 :

- **First**, take the two chromosomes associated with the same environment, one from each parent, and cut them both at the same fixed point, located right after the last node of the first hidden layer and right before the first node of the second hidden layer.
- **Second**, exchange the two resulting segments, which produces two recombinant chromosomes, one carrying the first segment of parent A with the second segment of parent B, and the other carrying the complementary combination.
- **Third**, select one of the two recombinants at random and assign it to the offspring at the position corresponding to that environment.

The offspring therefore receives one first-environment chromosome and one second-environment chromosome, each of which combines material from both parents.

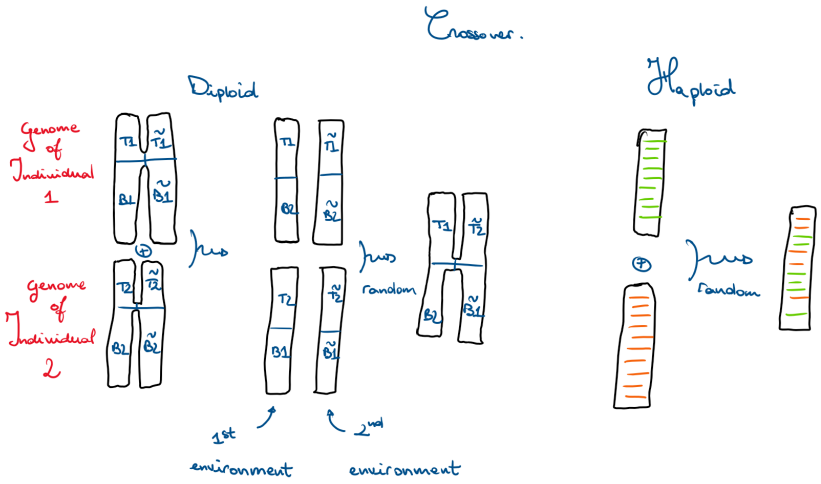


Figure 2.10: Chromosome-Level Plasticity and crossover between individuals

Mutation follows the exact same rules as in the developmental plasticity process. Every type of information has a chance to mutate, and this applies to both chromosomes regardless of the current environment. The chromosome that is not expressed in the current environment is therefore modified by crossover and mutation just like the expressed one, while being invisible to the selection process : it is inherited based on the fitness of the expressed chromosome rather than on its own, and may drift away from the solution it encoded when it was last expressed.

2.7 Chromosome-level plasticity - Haploid Neural Network

This version of the genome representation is the simplest one. The haploid genome is composed of a single chromosome, which is expressed in both environments. As in the diploid version, the input of the body ANN does not change either and remains a fixed bias. An individual therefore expresses a single phenotype, which is used in both environments.

Chapter 3

Results

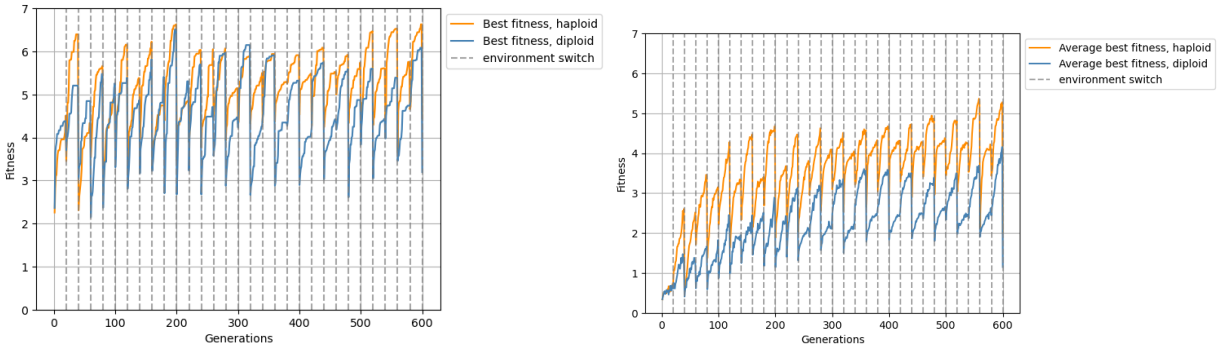
This chapter presents the results of the experiments described earlier. All the results reported here are averaged over 5 independent runs. Two pairs of environments are studied, Pusher-v0 with Walker-v0 and Thrower-v0 with Pusher-v0, and each pair is evolved under the two plasticity conditions, which gives four experiments in total.

Every experiment is reported in the same order : the fitness curves across generations, the violin plots at the end of each episode, the violin plots at the beginning of each episode, the percentage of invalid individuals, and finally a deeper analysis of the phenotypes. The Pusher-v0 and Walker-v0 pair is discussed in full under both plasticity conditions, since the two conditions lead to opposite results. The Thrower-v0 and Pusher-v0 pair is then used to check whether the same behaviour appears on a different pair of tasks : we report what we observe on the same figures and refer back to the corresponding discussion instead of repeating it.

3.1 Pusher and Walker

3.1.1 Developmental Plasticity

Figure 3.1 shows how the diploid and haploid versions performed on the alternating Pusher-v0 and Walker-v0 environments. The two environments always alternate in the order given by the title of the section : the first twenty generations are evaluated on Pusher-v0, the next twenty on Walker-v0, and so on. Both curves follow a sawtooth pattern : fitness drops at each environment switch and recovers over the following generations. We also observe on Figure 3.1 that the average fitness of the 40 best individuals globally increases over the run, which suggests that the population gradually improves its solutions for both environments. Overall, the haploid version appears to be more effective in these changing environments.



(a) Evolution across generations of the fitness of the best individual

(b) Evolution across generations of the average fitness of the 40 best individuals

Figure 3.1: Evolution of the two types of fitness.

On Walker-v0, our approach reaches high fitness values, comparable to those reported by Tanaka and Aranha [4]. Pusher-v0 proved harder, so we set the episode length to 1000 time steps to give the robots more time to succeed. A closer inspection of the recorded videos nevertheless shows that the algorithm found good designs, together with controllers acting in accordance with them. Another interesting point on Figure 3.1 is the rise in the average fitness of the 40 best individuals after each environment switch. This suggests that a single genome encodes a solution for both tasks, a point that the morphological analysis below examines further. This is the main result of our study : a single genome can encode not only one body and one controller, but two bodies and two controllers.

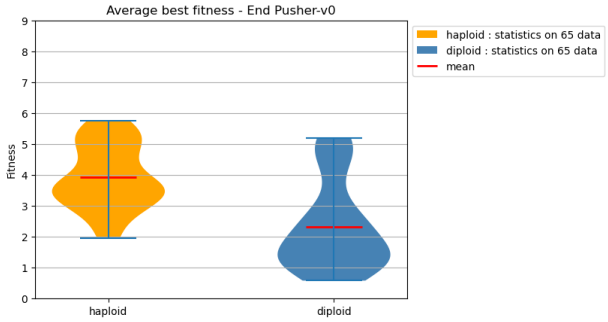
The other main result of this study is that the diploid algorithm does not perform as well as the haploid one. We suggest that this comes from both the body and the controller being indirectly encoded. Indirect encodings are known to suffer from low locality [12] [1] : because the genotype-phenotype mapping is highly non-trivial, small variations in the genotype can translate into large variations in the phenotype, so that even close genomes may decode to significantly different bodies and controllers.

On top of this, diploidy adds a further source of noise. A mutation can occur on a recessive allele : in that case it is not expressed and drifts silently in the population, diverging from the rest of the genome. When that allele later becomes dominant, the accumulated change is expressed at once and causes a much larger perturbation, so that what was meant to bring diversity brings noise instead. This masking effect may be a key reason why the haploid algorithm succeeds more often : in the haploid case every mutation is expressed and evaluated immediately, so a harmful one is selected against and removed straight away, whereas diploidy delays this evaluation and lets deleterious variation build up unseen.

To further understand the global performance of each algorithm, we propose the violin plots of Figure 3.2 and Figure 3.3, which focus on the critical moments of the experi-

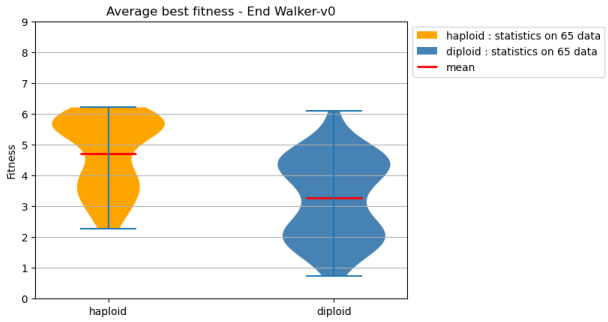
ment : the fitness at the end of each environment and the fitness at the beginning of each environment, both for the best individual and for the average fitness of the 40 best individuals. The central line represents the mean over the data extracted across generations and runs, while the top and bottom horizontal lines represent the maximum and the minimum reached. The statistics are computed after the 99th generation, so that the initial learning phase of the population does not impact the results.

(a) Best fitness at the end of each Pusher-v0 episode



(b) Average fitness of the 40 best individuals at the end of each Pusher-v0 episode

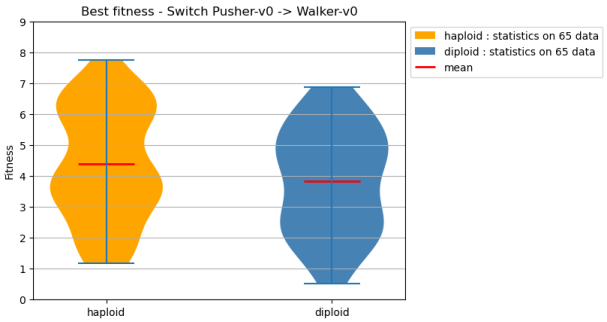
(c) Best fitness at the end of each Walker-v0 episode



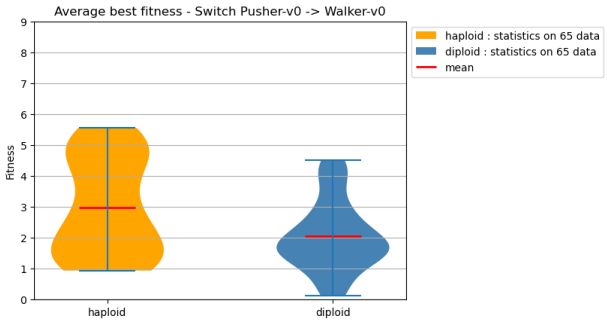
(d) Average fitness of the 40 best individuals at the end of each Walker-v0 episode

Figure 3.2: Violin plots of the fitness at the end of each episode, with upper and lower standard deviation.

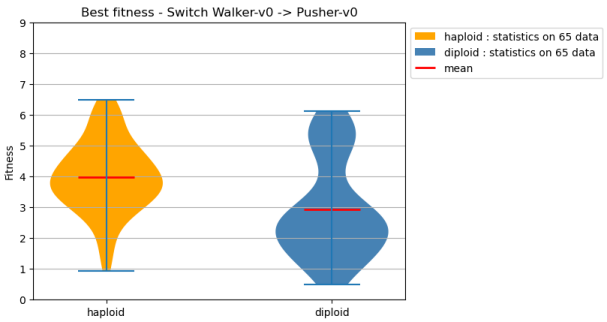
We observe that the haploid algorithm performs slightly better than the diploid one, since every colored part of the haploid violin lies above the diploid one. This holds at the end of each episode, on Figure 3.2, as well as at the first generation after a switch, on Figure 3.3.



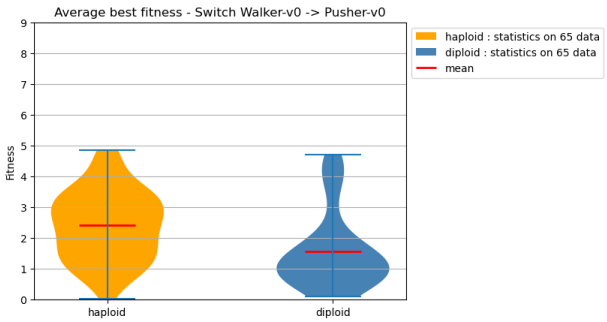
(a) Best fitness at the beginning of each Walker-v0 episode



(b) Average fitness of the 40 best individuals at the beginning of each Walker-v0 episode



(c) Best fitness at the beginning of each Pusher-v0 episode



(d) Average fitness of the 40 best individuals at the beginning of each Pusher-v0 episode

Figure 3.3: Violin plots of the fitness at the beginning of each episode, with upper and lower standard deviation.

To compare the two algorithms even further, Figure 3.4 shows the evolution of the percentage of invalid individuals per generation. The body ANN can generate bodies that are cut in two, and therefore invalid, or bodies that are too small, which are also considered invalid. This lets us see which algorithm has overfitted its solution to a given environment. Figure 3.4 shows that, overall, the haploid population produces fewer invalid individuals than the diploid one. This is consistent with the idea that diploidy brings in more unexpressed variation, which later shows up as unstable and invalid phenotypes rather than as useful diversity.

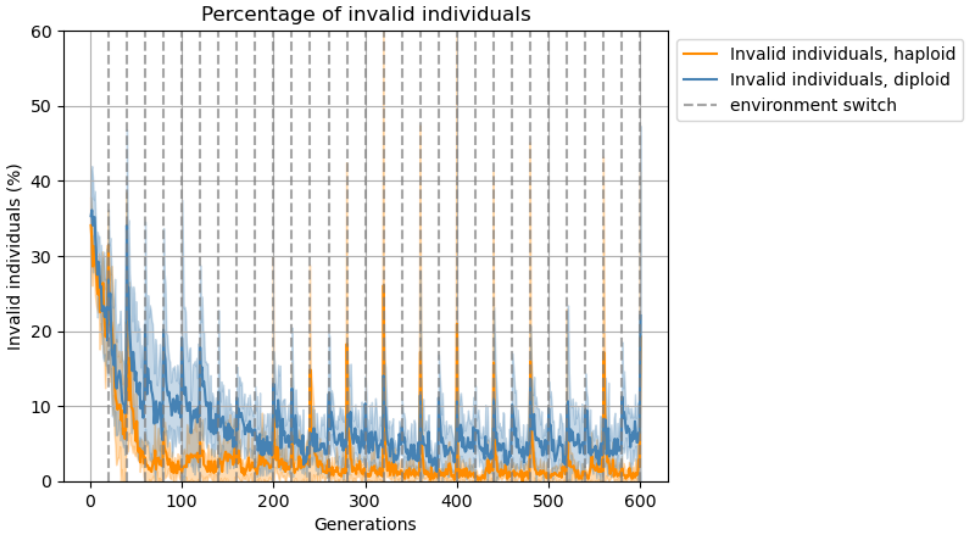


Figure 3.4: Percentage of invalid individuals per generation.

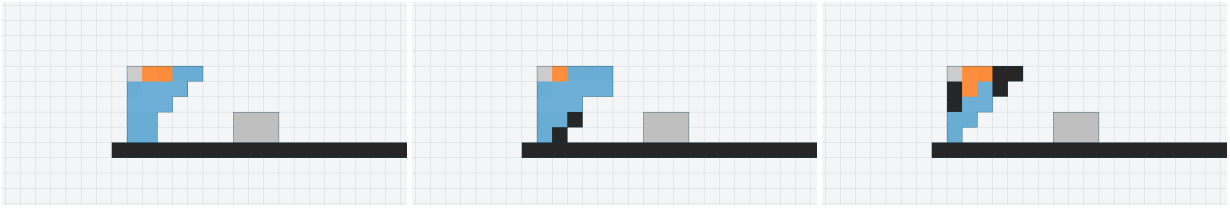
3.1.2 Developmental Plasticity - Deeper Analysis of the Results

To better understand the behavior of the algorithm, we rendered some of the best individuals. A playlist gathering a few good individuals for each task and each ploidy is available here. On other runs, different solutions that also reach good fitness values were found, some of them are shown on Figure 3.7 and Figure 3.8.

These videos show that the algorithm manages to find effective solutions. Interestingly, similar solutions can be found for both types of ploidy, which indicates that both algorithms can reach similar solutions. This suggests that the difference in the measured results is linked to the representation of the genome. For the Pusher-v0 environment, the best solutions are robots made of a mix of vertical and horizontal actuators : the front part pushes the box while the rear part acts as a single leg reinforced by the horizontal actuators. A solution with only vertical actuators was not found, and the presence of these horizontal voxels seems to be essential for this type of morphology to perform in this environment. On

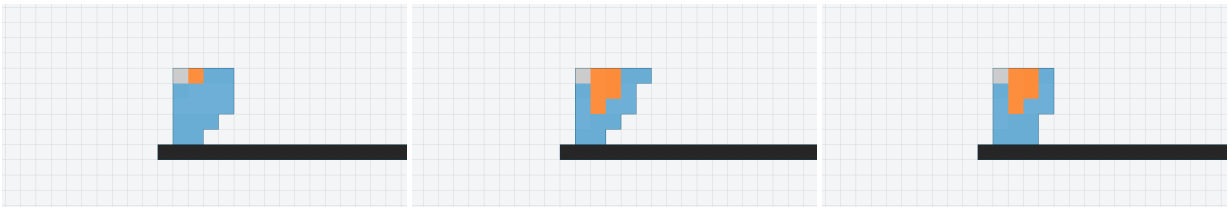
the Walker-v0 environment, the robots have more blocks, which results in better propulsion. Here again the horizontal voxels help the robot move forward, through the small rotation of the rear leg that results from their extension. One particularly interesting point is the almost systematic absence of a voxel in the lower-right part of the robot, which lets it lean forward and move.

Another point addressed by this analysis is whether the algorithm produces, along the genealogy, individuals that look alike, meaning that they share a similar phenotype even with an indirect encoding that suffers from low locality. Similar characteristics are indeed observed between individuals of the same family for both types of ploidy, as shown on Figure 3.5 and Figure 3.6. This suggests that the two algorithms and the current encoding allow inherited features to be passed on across generations. It also suggests that, even with an indirect encoding, crossover and mutation manage to produce good individuals, and that close genomes can produce similar phenotypic characteristics.



(a) Child of the family (b) Parent of the family (c) The other parent of the family

Figure 3.5: Example of the phenotype of a haploid family.



(a) Child of the family (b) Parent of the family (c) The other parent of the family

Figure 3.6: Example of the phenotype of a diploid family.

To analyze the results of the algorithm in more depth, we take the CPPN of the top n individuals right after the environment switch and, for each of them :

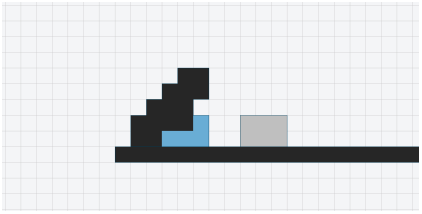
1. rebuild the body of the individual after the switch;
2. build the body of the individual as if it were in the environment before the switch;
3. compare the two bodies to assess the impact of the switch on the body;

4. compare the reconstructed body to the real bodies of the top n individuals present right before the switch, to see whether they are alike;

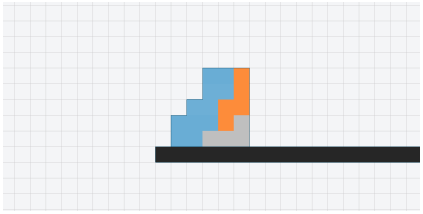
5. compare the body of the individual after the switch to the former top n bodies of this environment in the previous episode, to look for any kind of memory.

The goal is to determine whether alternative solutions are stored in the genome and reactivated, or whether a single genome carries both solutions and generalizes over the problem at hand. We study only the top n individuals because, if a past feature is useful, its reappearance should give a strong advantage to the individuals carrying it and place them among the best of their generation.

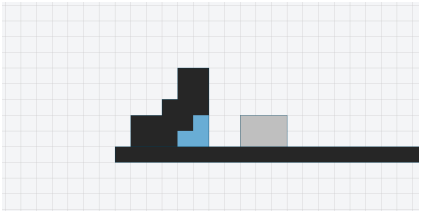
As seen on Figure 3.7 and Figure 3.8, the algorithm tends to show a generalization effect for both the haploid and the diploid version : the same body, in an altered form, comes back as a top individual after an episode, which suggests that the least modified genomes are able to keep a phenotype similar to that of the previous episode. Here, the Pusher-v0 robot has a fitness of 4 right after the switch and 9 at the last episode, and the Walker-v0 robot has a fitness of 5.1. These are good fitness values : not the highest the algorithm can reach, but solid ones.



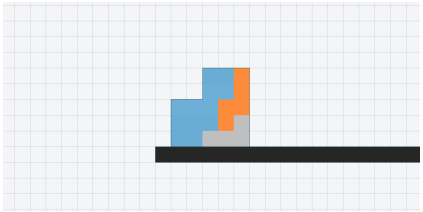
(a) Body created by a CPPN on haploid Pusher-v0 right after the switch



(b) Body created with the same CPPN on haploid Walker-v0

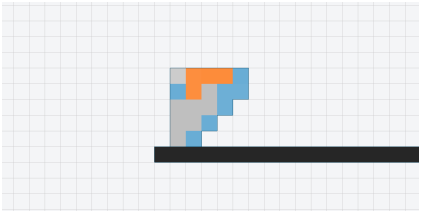


(c) Typical body of haploid Pusher-v0 before the episode of Walker-v0

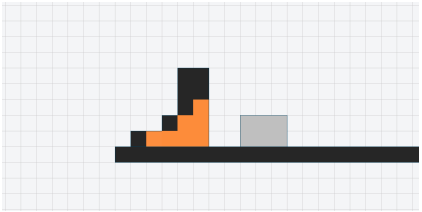


(d) Typical body on haploid Walker-v0 before the switch

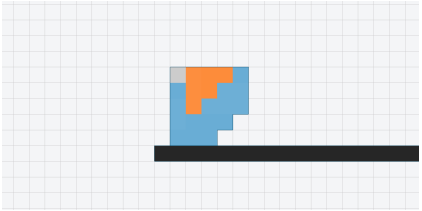
Figure 3.7: A deeper analysis of the haploid algorithm at the environment switch.



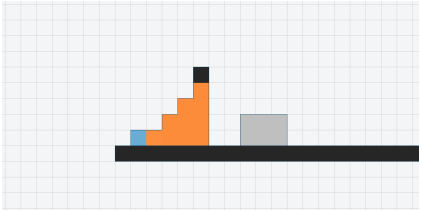
(a) Body created by a CPPN on diploid Walker-v0 right after the switch



(b) Body created with the same CPPN on diploid Pusher-v0



(c) Typical body of diploid Walker-v0 before the episode of Pusher-v0



(d) Typical body on diploid Pusher-v0 before the switch

Figure 3.8: A deeper analysis of the diploid algorithm at the environment switch.

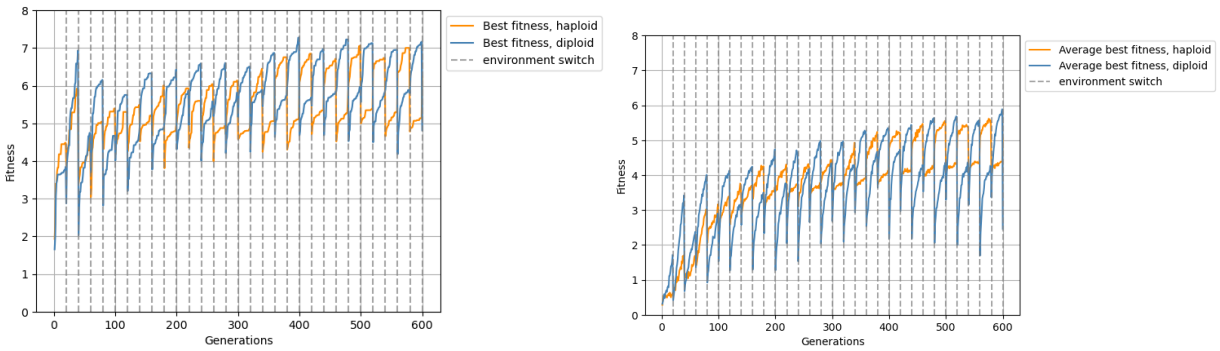
These robots show that phenotypic plasticity plays an important role in the evolution of the population : a single genome is able to produce two different pairs of body and controller for the two environments. Since the two algorithms share the same overall behavior, comparing them isolates the effect of diploidy itself : this suggests that the difference in performance may come from the additional variation introduced by the diploid representation, which does not benefit an indirect encoding that already suffers from low locality [12] [1]. Under developmental plasticity, diploidy appears to act mainly as a source of noise.

3.1.3 Chromosome-Level Plasticity

We now turn to the second plasticity condition on the same pair of environments. The genome is the same, but the environmental cue no longer enters the body ANN : the input is a fixed bias, and the cue decides which chromosome is expressed. The diploid individual therefore carries one complete CPPN per environment, while the haploid individual carries a single one that has to answer for both tasks. The results are reversed compared to the developmental condition.

Figure 3.9 compares, as before, the fitness reached by each of the two algorithms. We observe that the haploid algorithm has specialized on the Pusher-v0 environment and scores very poorly on Walker-v0. The diploid algorithm, on the other hand, keeps a good fitness

across the two environments, and even approaches on Pusher-v0 the performance of the haploid algorithm that specialized on it. Overall, the diploid algorithm adapts better to the two environments.



(a) Evolution across generations of the fitness of the best individual

(b) Evolution across generations of the average fitness of the 40 best individuals

Figure 3.9: Evolution of the two types of fitness.

This better adaptation seems associated with the genotype representation itself, where a single chromosome corresponds to a single environment. Each chromosome is under selection only every other episode, and can therefore specialize on its own task instead of having to account for both at once. The haploid version does not have this possibility : with a single chromosome and a fixed input, it expresses one phenotype for the two environments. It ends up specializing on one of them, and once evolution has gone far enough in that direction, the population does not seem able to come back and adapt to the other task.

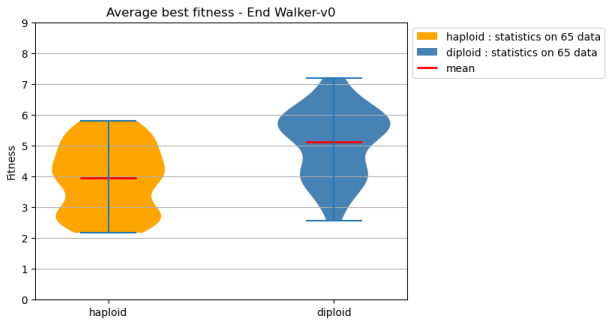
What makes this result less expected is that the chromosome which is not expressed keeps being modified by crossover and mutation while being invisible to selection : it is inherited according to the fitness of the expressed chromosome rather than on its own, and it may drift away from the solution it encoded when it was last expressed. Even with this drift, the diploid population recovers after each switch, which suggests that twenty generations are not enough to destroy the solution stored on the silent chromosome.

To further understand the global performance of each algorithm, Figure 3.10 and Figure 3.11 show the same violin plots as in the developmental plasticity condition, at the same critical moments of the experiment.

(a) Best fitness at the end of each Pusher-v0 episode

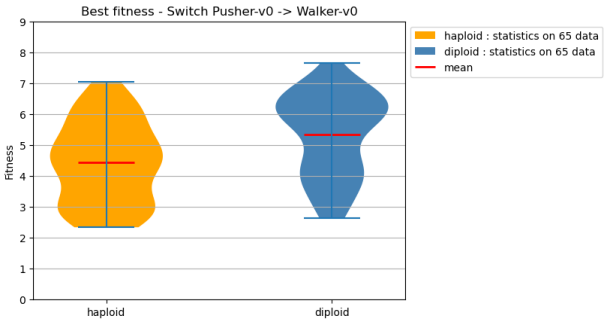
(b) Average fitness of the 40 best individuals at the end of each Pusher-v0 episode

(c) Best fitness at the end of each Walker-v0 episode

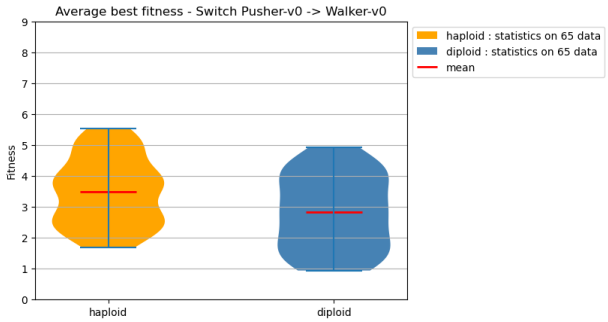


(d) Average fitness of the 40 best individuals at the end of each Walker-v0 episode

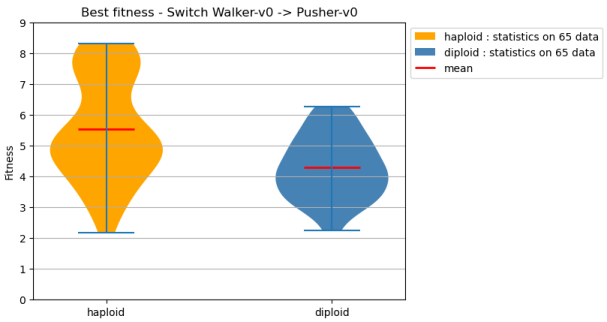
Figure 3.10: Violin plots of the fitness at the end of each episode, with upper and lower standard deviation.



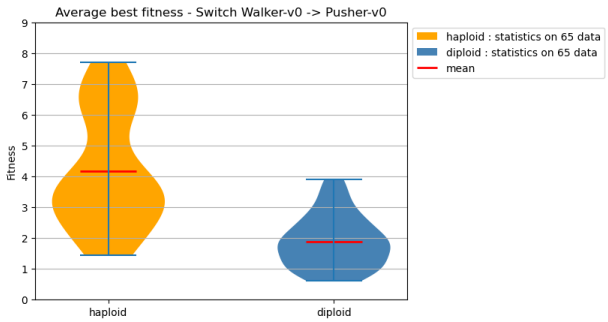
(a) Best fitness at the beginning of each Walker-v0 episode



(b) Average fitness of the 40 best individuals at the beginning of each Walker-v0 episode



(c) Best fitness at the beginning of each Pusher-v0 episode



(d) Average fitness of the 40 best individuals at the beginning of each Pusher-v0 episode

Figure 3.11: Violin plots of the fitness at the beginning of each episode, with upper and lower standard deviation.

On Figure 3.10 we observe that the haploid algorithm has specialized on the Pusher-v0 task, where it stays slightly ahead, while the diploid one performs well on both and takes the lead on Walker-v0. On Figure 3.11, at the first generation after a switch, the diploid algorithm is not always ahead : the drift of the silent chromosome costs it something at the exact moment of the switch, and this is the weakest point of this representation.

Read together with the fitness curves of Figure 3.9, these plots nevertheless suggest that the two representations behave differently after the switch.The fitness of the haploid algorithm stays flat over the episode, its genotype offering no room for improvement once it has specialized on the other task, whereas the fitness of the diploid algorithm rises quickly : even after twenty generations of drift, the expressed chromosome recovers and adapts to the current environment.

Figure 3.12 shows the evolution of the percentage of invalid individuals per generation. This time, the two algorithms show almost the same percentage, although the haploid curve stays below the diploid one. A possible reason is that the haploid algorithm is not forced to change its body at the switch, while the diploid one expresses a chromosome that has been drifting for a whole episode, which results in more variation and in more invalid bodies. The gap between the two curves is much smaller than under developmental plasticity, where the unexpressed variation was spread over the whole genome instead of being confined to one chromosome.

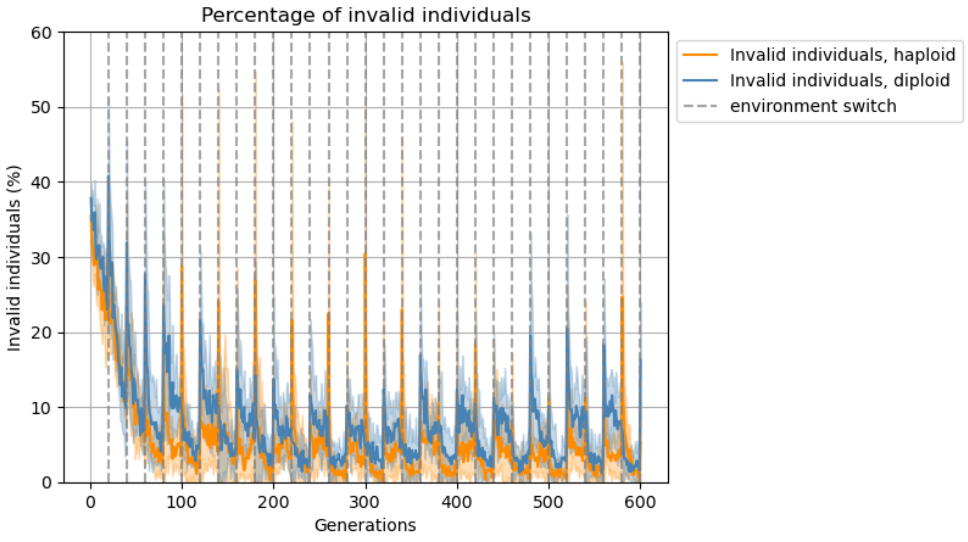


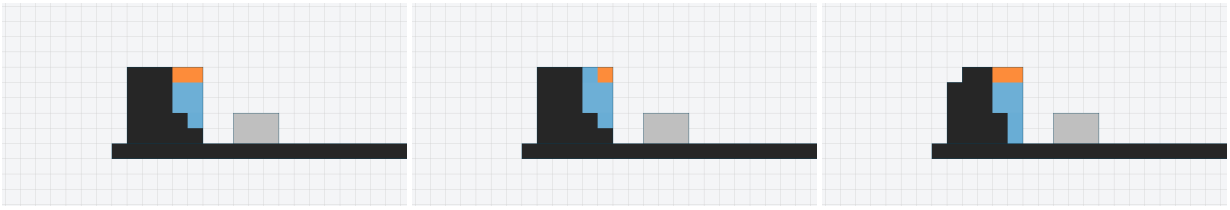
Figure 3.12: Percentage of invalid individuals per generation.

3.1.4 Chromosome-Level Plasticity - Deeper Analysis of the Results

To better understand the behavior of the algorithm, we rendered some of the best individuals. A playlist gathering a few good individuals for each task and each ploidy is available here.

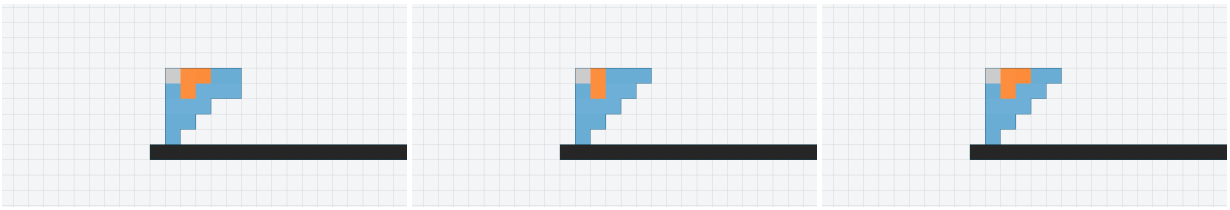
These videos show that the algorithm manages to find effective solutions. We observe that the diploid algorithm finds two of them, one per environment, which is what the fitness curves suggested. They also explain the poor results of the haploid algorithm on the Walker-v0 task : the robot behaves as if it were in the Pusher-v0 environment and cannot make the larger moves that would give it a better score. The haploid individual is not failing to control its body, it is controlling a body built for the other task.

Figure 3.13 and Figure 3.14 show an example of a haploid and a diploid family. As in the developmental condition, individuals of the same family share visible characteristics, which suggests that the crossover applied chromosome by chromosome transmits inherited features just as well.



(a) Child of the family (b) Parent of the family (c) The other parent of the family

Figure 3.13: Example of the phenotype of a haploid family.

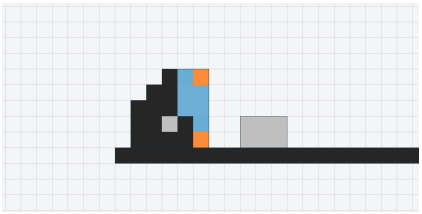


(a) Child of the family (b) Parent of the family (c) The other parent of the family

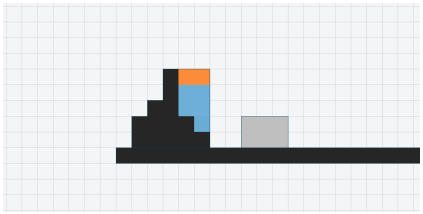
Figure 3.14: Example of the phenotype of a diploid family.

Following the same protocol as in Section 3.1.2, but this time switching the whole chromosome instead of the sign of the input of the body ANN, Figure 3.15 and Figure 3.16 compare the bodies before and after the environment switch. On Figure 3.15, the haploid body does not change at the switch, which is expected since a single chromosome

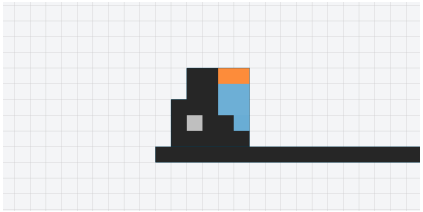
is expressed in both environments : we observe no adaptation to the new task. On Figure 3.16, the diploid algorithm shows a memorization of the past working solution, the body expressed after the switch being close to the one expressed during the previous episode of the same environment.



(a) Body created by a CPPN on haploid Pusher-v0 right after the switch

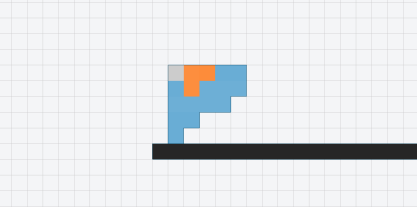


(b) Typical body of haploid Pusher-v0 before the episode of Walker-v0

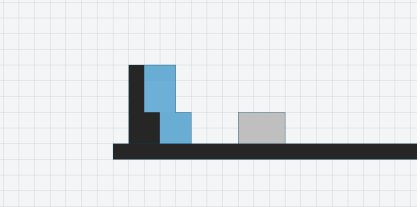


(c) Typical body on haploid Walker-v0 before the switch

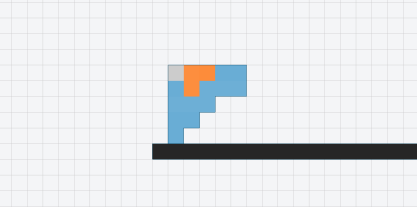
Figure 3.15: A deeper analysis of the haploid algorithm at the environment switch.



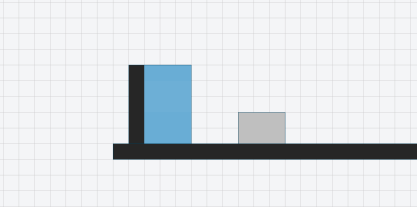
(a) Body created by a CPPN on diploid Walker-v0 right after the switch



(b) Body created with the other CPPN on diploid Pusher-v0



(c) Typical body of diploid Walker-v0 before the episode of Pusher-v0



(d) Typical body on diploid Pusher-v0 before the switch

Figure 3.16: A deeper analysis of the diploid algorithm at the environment switch.

These figures suggest that, even though crossover and mutation can damage the silent chromosome, some individuals still carry a working solution at the moment of the switch. Read together with the violin plots and the fitness curves, this indicates that a few individuals manage to produce a good robot right after the switch, and that these few are the ones that allow the population to recover quickly.

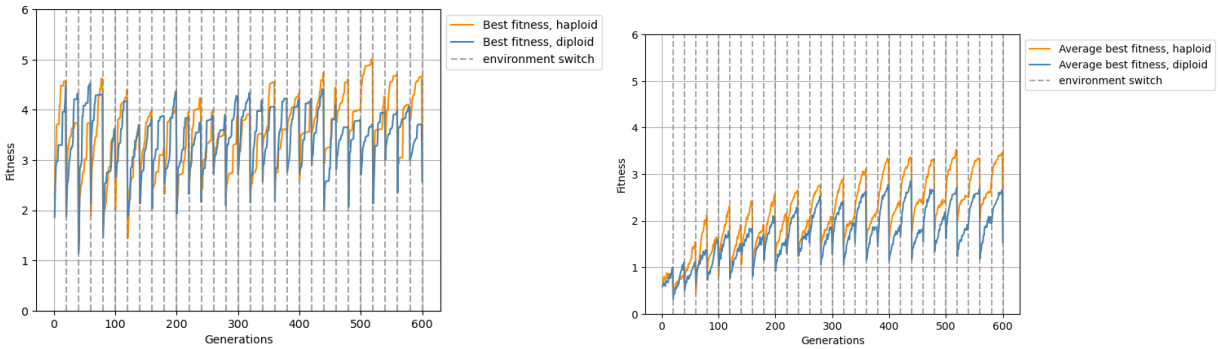
Under chromosome-level plasticity, the diploid representation is therefore the one that adapts to both environments, which is the opposite of what we observed under developmental plasticity. The two conditions are however not equivalent : here the haploid genome carries a single chromosome and a fixed input, and therefore has no mechanism of plasticity at all, while the diploid one carries a complete CPPN per task. The comparison in this condition measures the presence or the absence of plasticity as much as the ploidy of the genome, and the advantage we observe should be read in this light.

3.2 Thrower and Pusher

This second pair of environments is used to check whether the behaviour observed on Pusher-v0 and Walker-v0 also appears on different tasks. The figures are the same and follow the same order, so we report here what we observe and refer back to the discussion of Section 3.1.2 rather than repeating it.

3.2.1 Developmental Plasticity

Figure 3.17 displays the best fitness along with the average fitness of the 40 best individuals. For both algorithms, the 40 best individuals increase in capability before reaching what seems to be a stationary state, which suggests that the population progressively adapts to the two environments and establishes solutions for both with a single genome. Here again the haploid algorithm performs better, even though its advantage is less marked this time. Moreover, the best individuals on the Thrower-v0 task match the fitness of the Thrower-v0 robot reported by [4], and even seem to improve on it.

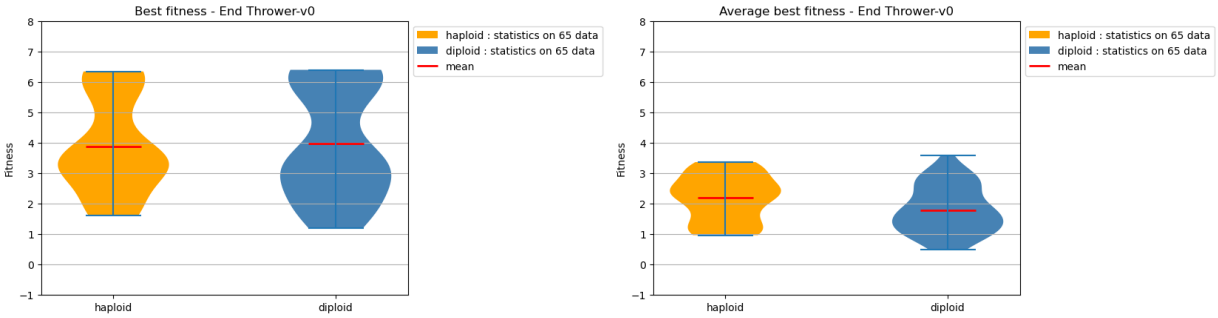


(a) Evolution across generations of the fitness of the best individual

(b) Evolution across generations of the average fitness of the 40 best individuals

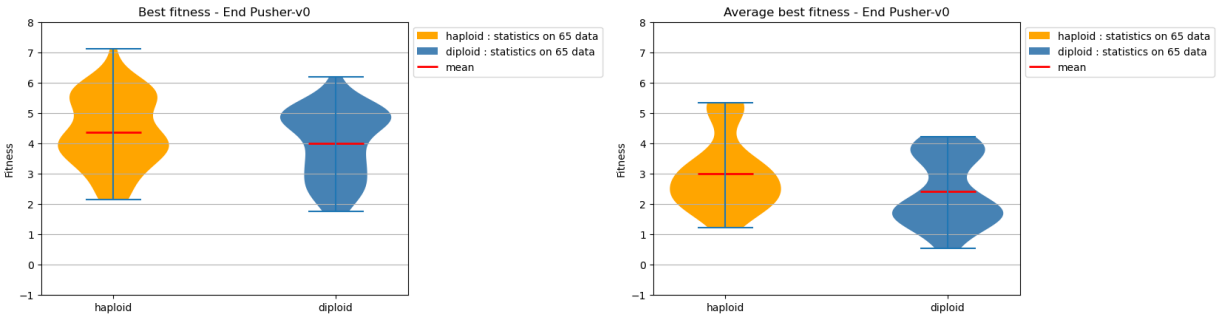
Figure 3.17: Evolution of the two types of fitness.

Figure 3.18 and Figure 3.19 show the violin plots at the end and at the beginning of each episode. On Figure 3.18 we can see the improvement of the Thrower-v0 robots compared to the results of Tanaka and Aranha, and it seems that this time the diploid algorithm catches up a little with the haploid one, the gap between the two being smaller than in the previous experiment. On Figure 3.19, at the first generation after a switch, the haploid algorithm keeps a small advantage on both tasks.



(a) Best fitness at the end of each Thrower-v0 episode

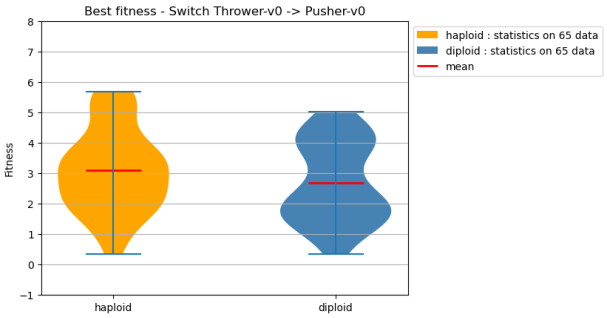
(b) Average fitness of the 40 best individuals at the end of each Thrower-v0 episode



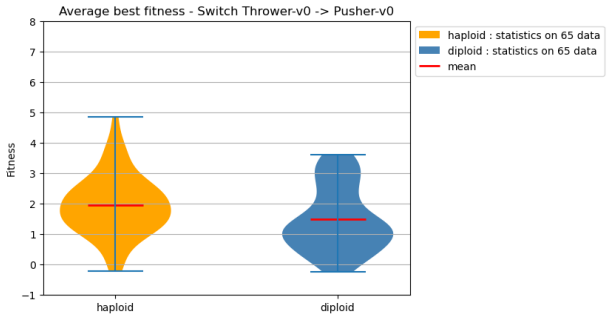
(c) Best fitness at the end of each Pusher-v0 episode

(d) Average fitness of the 40 best individuals at the end of each Pusher-v0 episode

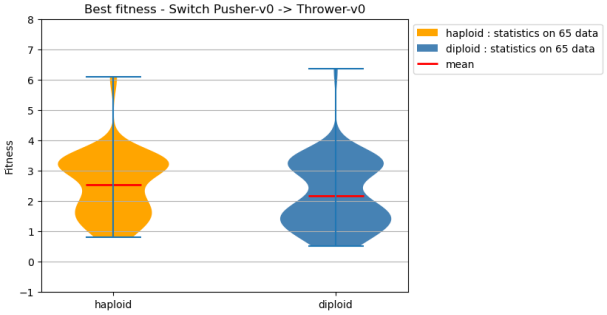
Figure 3.18: Violin plots of the fitness at the end of each episode, with upper and lower standard deviation.



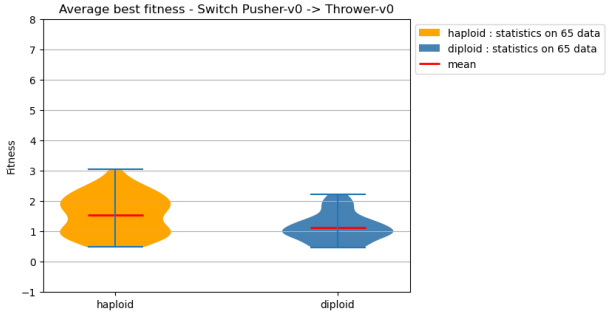
(a) Best fitness at the beginning of each Pusher-v0 episode



(b) Average fitness of the 40 best individuals at the beginning of each Pusher-v0 episode



(c) Best fitness at the beginning of each Thrower-v0 episode



(d) Average fitness of the 40 best individuals at the beginning of each Thrower-v0 episode

Figure 3.19: Violin plots of the fitness at the beginning of each episode, with upper and lower standard deviation.

Figure 3.20 shows the percentage of invalid individuals across generations, and this time the behavior differs from the previous experiment. At the environment switch, the diploid algorithm is now the less regular of the two, while the haploid one was the less regular in the Pusher and Walker experiment. Overall, the haploid algorithm still produces fewer invalid individuals than the diploid one.

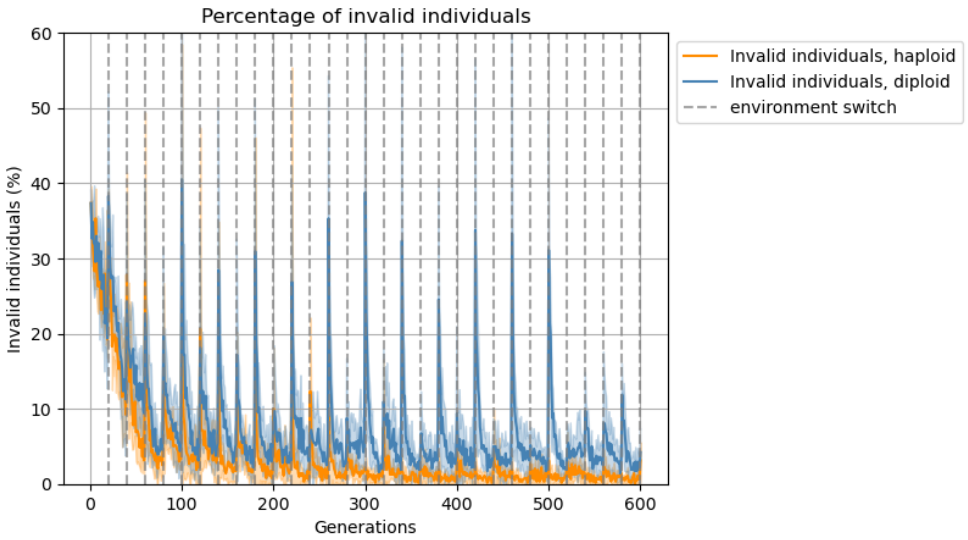
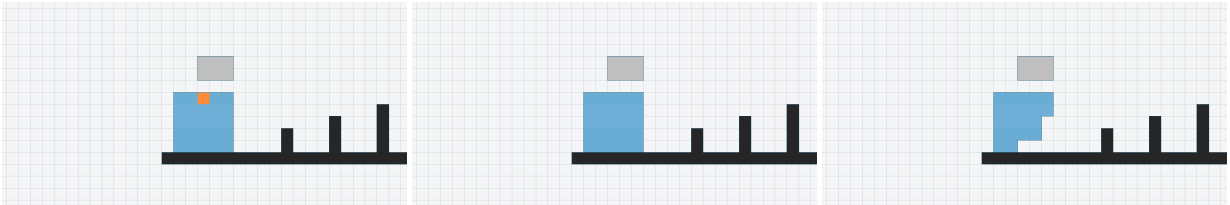


Figure 3.20: Percentage of invalid individuals per generation.

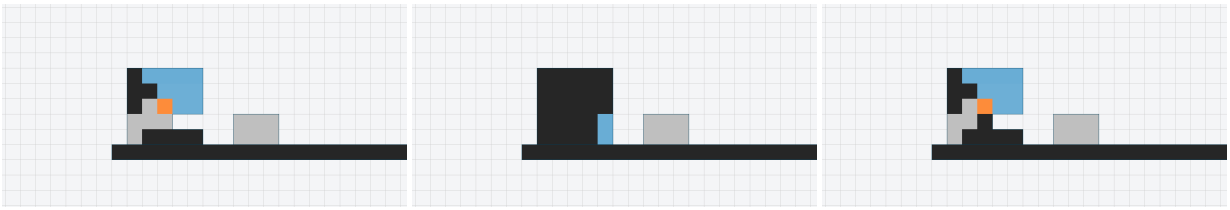
A playlist gathering a few good individuals for each task and each ploidy is available here. What is particularly interesting to notice is that, for the Thrower-v0 robots, the best performing individuals control the block in order to place it above their body, which lets them optimize the initial direction of the throw.

Figure 3.21 and Figure 3.22 show an example of a haploid and a diploid family, and Figure 3.23 and Figure 3.24 apply the protocol of Section 3.1.2 at the environment switch. We observe the same generalization effect as on the previous pair of environments, for both types of ploidy, and the same resemblance between individuals of a same family.



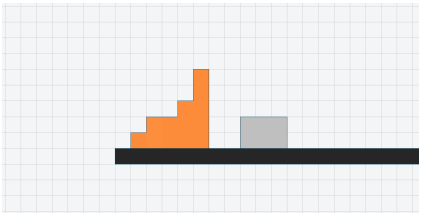
(a) Child of the family (b) Parent of the family (c) The other parent of the family

Figure 3.21: Example of the phenotype of a haploid family.

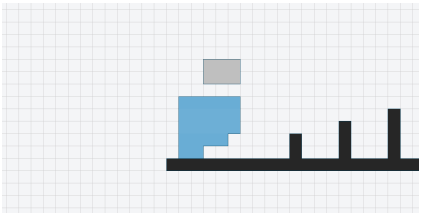


(a) Child of the family (b) Parent of the family (c) The other parent of the family

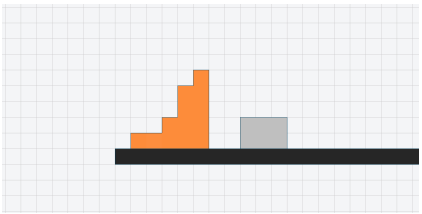
Figure 3.22: Example of the phenotype of a diploid family.



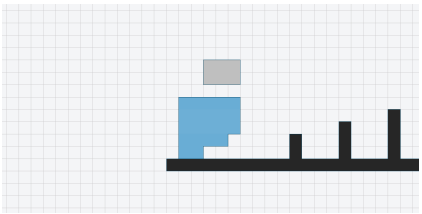
(a) Body created by a CPPN on haploid Pusher-v0 right after the switch



(b) Body created with the same CPPN on haploid Thrower-v0

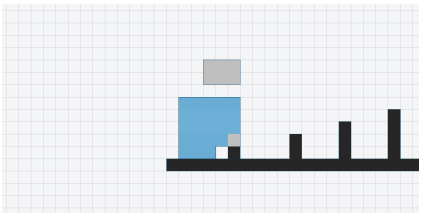


(c) Typical body of haploid Pusher-v0 before the episode of Thrower-v0

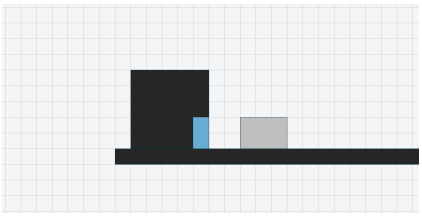


(d) Typical body on haploid Thrower-v0 before the switch

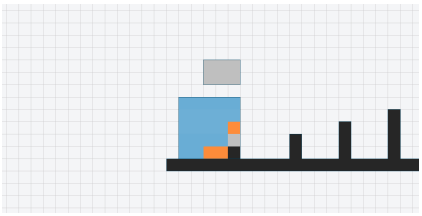
Figure 3.23: A deeper analysis of the haploid algorithm at the environment switch.



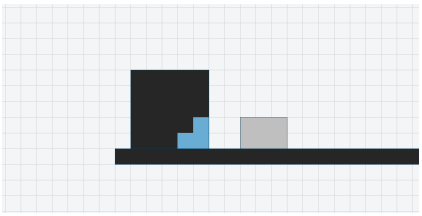
(a) Body created by a CPPN on diploid Thrower-v0 right after the switch



(b) Body created with the same CPPN on diploid Pusher-v0



(c) Typical body of diploid Thrower-v0 before the episode of Pusher-v0

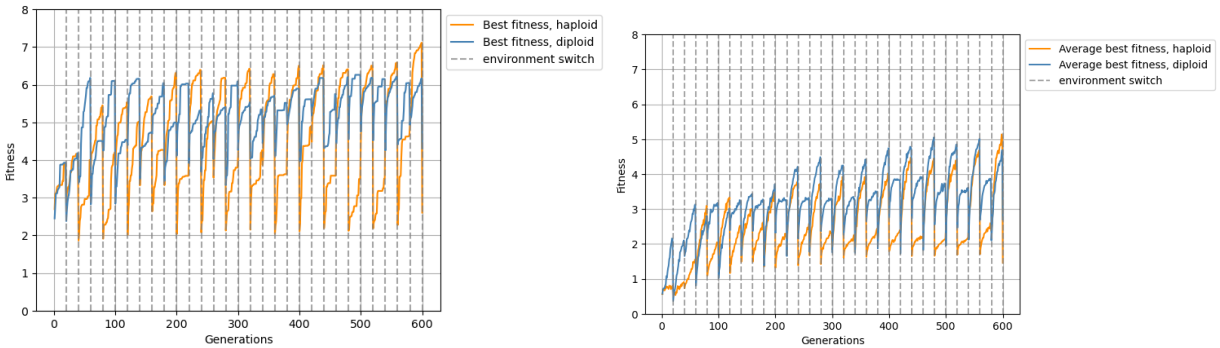


(d) Typical body on diploid Pusher-v0 before the switch

Figure 3.24: A deeper analysis of the diploid algorithm at the environment switch.

3.2.2 Chromosome-Level Plasticity

Figure 3.25 displays the best fitness along with the average fitness of the 40 best individuals. We observe the same reversal as on the previous pair of environments : the diploid algorithm now dominates the haploid one, which suggests that each of its chromosomes has adapted to its own environment. Moreover, the best diploid individuals on the Thrower-v0 task systematically outperform the fitness of the Thrower-v0 robot reported by [4].



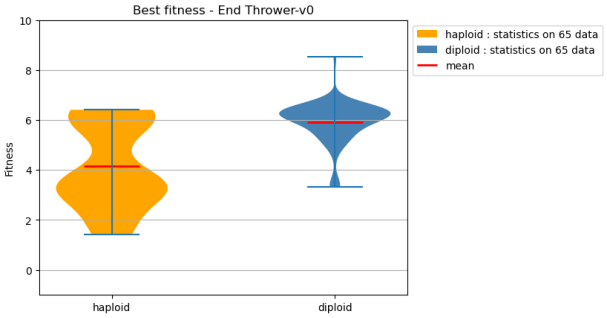
(a) Evolution across generations of the fitness of the best individual

(b) Evolution across generations of the average fitness of the 40 best individuals

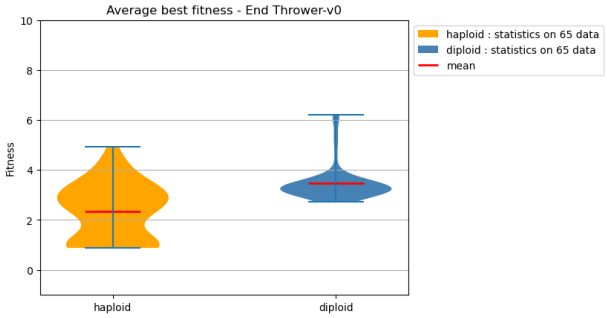
Figure 3.25: Evolution of the two types of fitness.

On Figure 3.26 and Figure 3.27, the two representations either perform the same or

the diploid one performs better. These plots also show again the weak point of the diploid genome : the chromosome that was not expressed drifted for twenty generations, and when it comes back it does not offer the same quality of phenotype as before. We observe nonetheless that the phenotype did not change too much, and that after a few generations the individuals in the population recover their past working solution, which suggests that the good genome has not disappeared but has only been weakened.

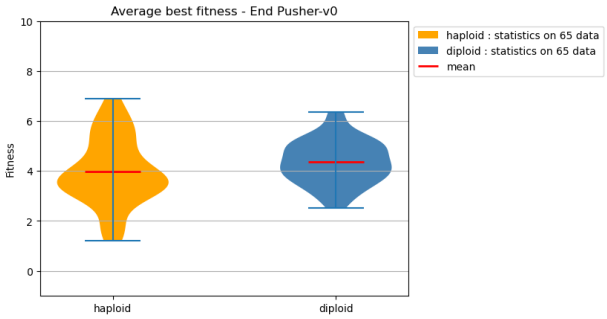


(a) Best fitness at the end of each Thrower-v0 episode



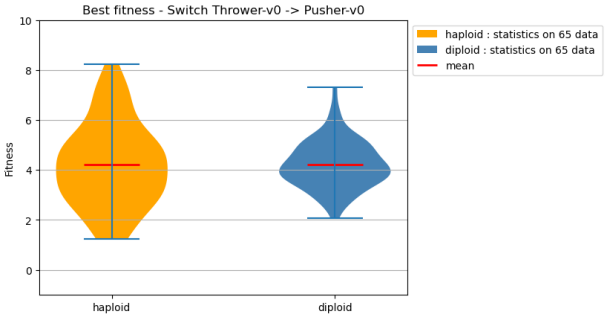
(b) Average fitness of the 40 best individuals at the end of each Thrower-v0 episode

(c) Best fitness at the end of each Pusher-v0 episode

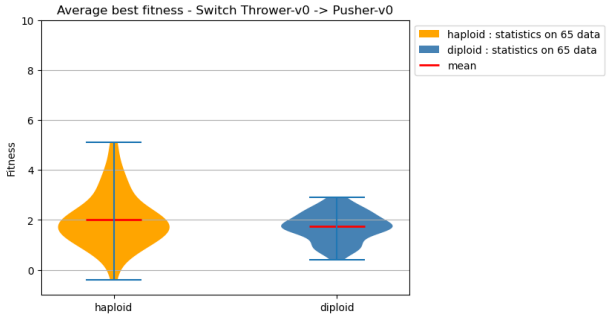


(d) Average fitness of the 40 best individuals at the end of each Pusher-v0 episode

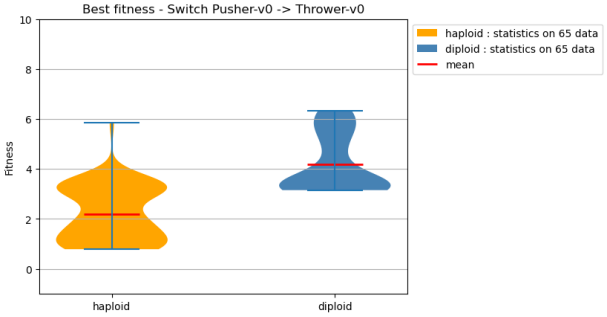
Figure 3.26: Violin plots of the fitness at the end of each episode, with upper and lower standard deviation.



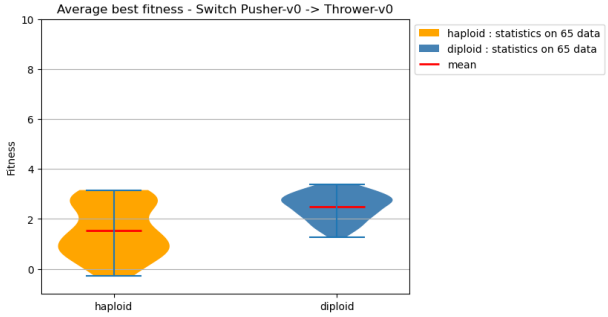
(a) Best fitness at the beginning of each Pusher-v0 episode



(b) Average fitness of the 40 best individuals at the beginning of each Pusher-v0 episode



(c) Best fitness at the beginning of each Thrower-v0 episode



(d) Average fitness of the 40 best individuals at the beginning of each Thrower-v0 episode

Figure 3.27: Violin plots of the fitness at the beginning of each episode, with upper and lower standard deviation.

Figure 3.28 shows the percentage of invalid individuals across generations. It is now unclear which representation produces fewer invalid individuals overall, but at the switch the diploid one is the one that produces the most. We again associate this with the twenty generations during which the expressed chromosome had been drifting.

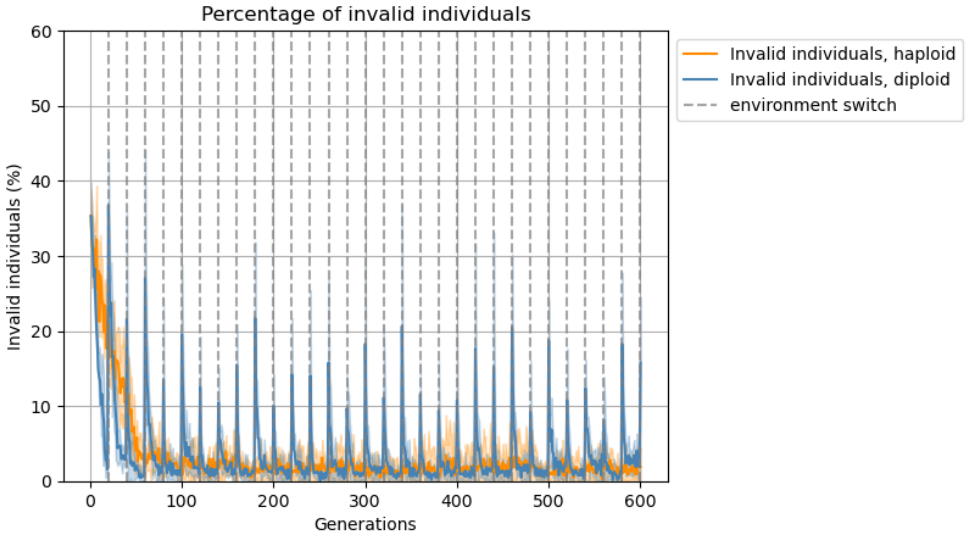
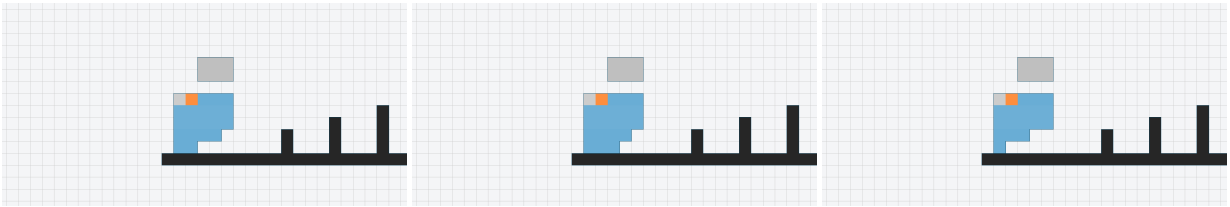


Figure 3.28: Percentage of invalid individuals per generation.

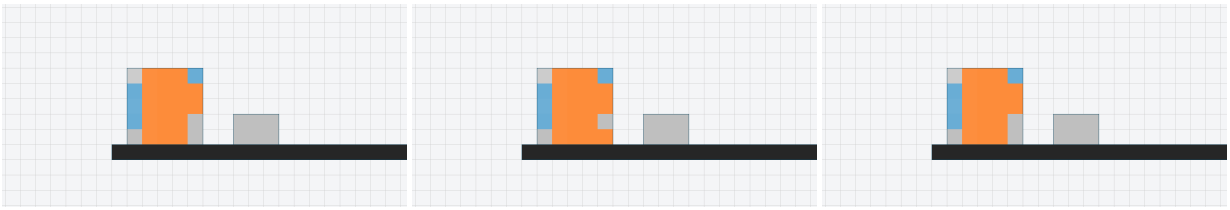
A playlist gathering a few good individuals for each task and each ploidy is available here. Here again, the best performing Thrower-v0 robots control the block in order to place it above their body before the throw.

Figure 3.29 and Figure 3.30 show an example of a haploid and a diploid family, and Figure 3.31 and Figure 3.32 apply the protocol of Section 3.1.2 at the environment switch. The observations are the same as on the previous pair of environments : the haploid body does not change at the switch, while the diploid one keeps a phenotype close to the one expressed during the previous episode of the same environment.



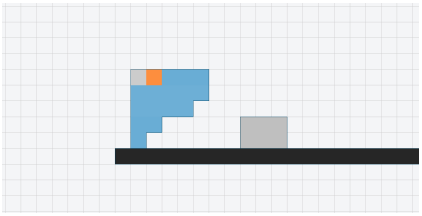
(a) Child of the family (b) Parent of the family (c) The other parent of the family

Figure 3.29: Example of the phenotype of a haploid family.

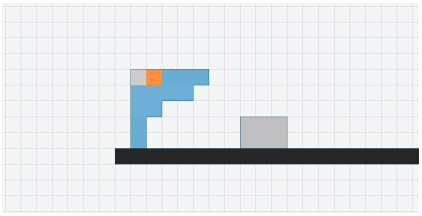


(a) Child of the family (b) Parent of the family (c) The other parent of the family

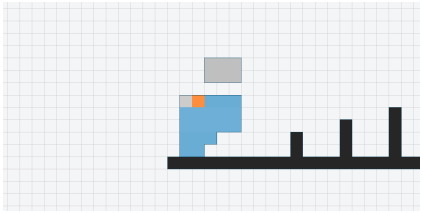
Figure 3.30: Example of the phenotype of a diploid family.



(a) Body created by a CPPN on haploid Pusher-v0 right after the switch

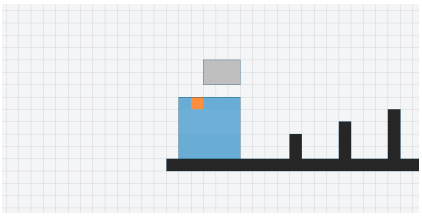


(b) Typical body of haploid Pusher-v0 before the episode of Thrower-v0

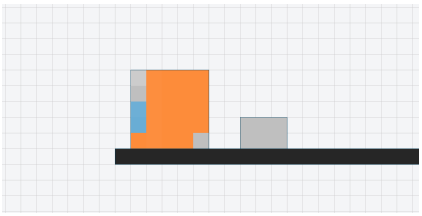


(c) Typical body on haploid Thrower-v0 before the switch

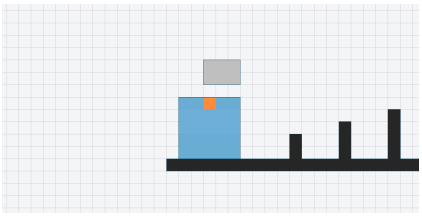
Figure 3.31: A deeper analysis of the haploid algorithm at the environment switch.



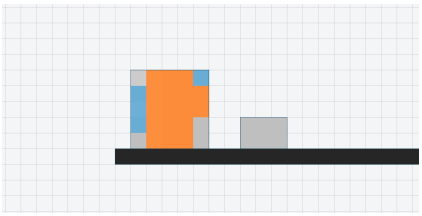
(a) Body created by a CPPN on diploid Thrower-v0 right after the switch



(b) Body created with the other CPPN on diploid Pusher-v0



(c) Typical body of diploid Thrower-v0 before the episode of Pusher-v0



(d) Typical body on diploid Pusher-v0 before the switch

Figure 3.32: A deeper analysis of the diploid algorithm at the environment switch.

Across both pairs of environments, the two plasticity conditions therefore lead to the same contrast : diploidy is a source of noise when the same genetic material has to be expressed in both environments, and an advantage when each chromosome can specialize on one of them.

Conclusion

This work started from the single genome formulation of Tanaka and Aranha [4], in which one CPPN generates both the body ANN and the controller ANN of a soft robot, and extended it in two directions. The first one is the environment : two tasks alternate at a fixed frequency during a single run, so that the same genome must account for two bodies and two controllers. The second one is the genome itself, converted into a diploid form where every gene is a whole node of the CPPN, expressed node by node through a dominance value and recombined by a meiosis-like crossover. Phenotypic plasticity was implemented at two levels : developmental, where the same genetic material is expressed in both environments and a single cue given as an input to the body ANN carries the environmental change, and chromosome-level, where each chromosome is dedicated to one environment and the cue decides which one is expressed.

Our first result is that this formulation holds : a single genome can encode not only one body ANN and one controller ANN, but two of each. On both pairs of environments the average fitness of the 40 best individuals increases over the run and rises again after each switch, and the rendered individuals show morphologies that differ between the two tasks. On Walker-v0 we reach fitness values comparable to those reported by Tanaka and Aranha, and on Thrower-v0 the best individuals match their results and seem to improve on them.

Our second result goes against our initial expectation. Under developmental plasticity the diploid representation does not perform as well as the haploid one : it reaches lower fitness at the end of each episode and produces more invalid individuals throughout the run. We suggest that diploidy adds a source of noise here. A mutation carried by a recessive allele is not expressed and drifts silently in the population, and when that allele later becomes dominant, the accumulated change is expressed at once.

Under chromosome-level plasticity the result is reversed. The diploid representation is the one that adapts to both environments, while the haploid one specializes on Pusher-v0 and behaves on Walker-v0 exactly as it does in the first task. The two conditions are however not equivalent : in the second one, the haploid genome carries a single chromosome and a fixed input, and therefore has no mechanism of plasticity at all.

Several limitations should be kept in mind. The topology of the CPPN is fixed and

the genetic algorithm deliberately simple. Only two pairs of environments were studied, at a single switching frequency, whereas the frequency of the change is itself reported to be a determining factor [7]. Varying this frequency would be the most direct continuation of this work, in order to find the regime in which the memory carried by the unexpressed chromosome becomes an advantage rather than a source of noise. Studying the mutation rate of the dominance on its own, or lowering the mutation rate of the unexpressed chromosome so that it keeps the solution it encoded when it was last expressed, would be two other ways to continue.

Bibliography

[1] Federico Pigozzi, Federico Julian Camerota Verdù, and Eric Medvet. How the morphology encoding influences the learning ability in body-brain co-optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1045–1054, 2023.

[2] Jagdeep Bhatia, Holly Jackson, Yunsheng Tian, Jie Xu, and Wojciech Matusik. Evolution gym: A large-scale benchmark for evolving soft robots. *Advances in Neural Information Processing Systems*, 34:2201–2214, 2021.

[3] Federico Pigozzi, Yujin Tang, Eric Medvet, and David Ha. Evolving modular soft robots without explicit inter-module communication using local self-attention. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 148–157, 2022.

[4] Fabio Tanaka and Claus Aranha. Co-evolving morphology and control of soft robots using a single genome. In *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1235–1242. IEEE, 2022.

[5] Nicholas Cheney, Josh Bongard, Vytas SunSpiral, and Hod Lipson. On the difficulty of co-optimizing morphology and control in evolved virtual creatures. In *Artificial life conference proceedings*, pages 226–233. MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info . . . , 2016.

[6] Raffaele Calabretta, Riccardo Galbiati, Stefano Nolfi, and Domenico Parisi. Two is better than one: a diploid genotype for neural networks. *Neural Processing Letters*, 4(3):149–155, 1996.

[7] Raffaele Calabretta, Stefano Nolfi, Domenico Parisi, and Riccardo Galbiati. Diploid robots adapting to fast changing environments. In *International Conference on Artificial Neural Networks*, pages 1145–1150. Springer, 1998.

[8] Cara Reedy. Diploidy for evolving neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1918–1921, 2018.

[9] Ralf J Sommer. Phenotypic plasticity: From theory and genetics to current and future challenges. *Genetics*, 215(1):1–13, 05 2020.

[10] Kenneth O Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007.

[11] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

[12] Franz Rothlauf. On the locality of representations. *None*, 2003.

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

[14] Kazuaki Harada and Hitoshi Iba. Lamarckian co-design of soft robots via transfer learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 832–840, 2024.

[15] Jozef Fekiač, Ivan Zelinka, and Juan C Burguillo. A review of methods for encoding neural network topologies in evolutionary computation. In *Proceedings of 25th European conference on modeling and simulation ECMS*, pages 410–416, 2011.

[16] Paulo Amaral, Silvia Carbonell-Sala, Francisco M De La Vega, Tiago Faial, Adam Frankish, Thomas Gingeras, Roderic Guigo, Jennifer L Harrow, Artemis G Hatzigeorgiou, Rory Johnson, et al. The status of the human gene catalogue. *Nature*, 622(7981):41–47, 2023.

[17] Bente Pakkenberg, Dorte Pelvig, Lisbeth Marner, Mads J. Bundgaard, Hans Jørgen G. Gundersen, Jens R. Nyengaard, and Lisbeth Regeur. Aging and the human neocortex. *Experimental Gerontology*, 38(1):95–99, 2003. Proceedings of the 6th International Symposium on the Neurobiology and Neuroendocrinology of Aging.

[18] Sarah P. Otto and Aleeza C. Gerstein. The evolution of haploidy and diploidy. *Current Biology*, 18(24):R1121–R1124, 2008.

[19] Oliviu Matei, Rudolf Erdei, and Laura Andreica. Diploid genetic algorithms: A comprehensive survey of theory, applications, and future directions. *Applications, and Future Directions*, 2026.

[20] Kyle Pretorius and Nelishia Pillay. Neural network crossover in genetic algorithms using genetic programming. *Genetic Programming and Evolvable Machines*, 25(1):7, 2024.

Summary —

In this report we introduced a single genome, indirectly encoded representation in which one CPPN generates both the body ANN and the controller ANN of a soft robot, placed in a changing environment where two tasks alternate at a fixed frequency. We studied it along two dimensions. Phenotypic plasticity was implemented at two levels : developmental, where a single cue given as an input to the body ANN carries the environmental change while the genome is read the same way in both environments, and chromosome-level, where each chromosome is dedicated to one environment and the cue decides which one is expressed. Ploidy was studied by converting the genome into a diploid form, every gene being a whole node of the CPPN, expressed node by node through a dominance value and recombined by a meiosis-like crossover. Our first result is that a single genome can encode not only one body ANN and one controller ANN, but two of each. The second one goes against our initial expectation : under developmental plasticity the diploid representation does not perform as well as the haploid one, indirect encodings being known to suffer from low locality, on top of which diploidy adds a further source of noise. Under chromosome-level plasticity the result is reversed, the diploid representation being the one that adapts to both environments while the haploid one specializes on a single task.

Keywords : Voxel-based Soft Robots, co-evolution, indirect encoding, CPPN, diploidy, haploidy, changing environments, phenotypic plasticity.

ISAE
10, avenue Édouard Belin
BP 54032
31055 Toulouse CEDEX 4