Evolution of evolvability for neuroevolution

Adam Katona

Doctor of Philosophy

University of York

Computer Science

October 2023

Abstract

Scientists have been in awe of the powers of evolution since comprehending the fundamental principles of natural selection. As Darwin eloquently put it, nature has given rise to "endless forms most beautiful". However, beyond the mesmerizing beauty of nature, lies an even more fascinating aspect - the creation of human intelligence. Although AI research has produced impressive outcomes in the past decade, the techniques employed to accomplish these outcomes diverge considerably from the natural process of brain evolution. The primary dissimilarity stems from the fact that nature employs the evolution of DNA code, which encodes the instructions for brain development, whereas mainstream AI represents all brain parameters directly. This disparity is noteworthy as the indirect representation utilized by nature confers upon it a potent ability to enhance evolvability over time, a feature that is absent in direct representation. This potential stems from the ability to choose instructions in a manner that renders the brain resistant to change in certain directions while facilitating easy modification in other directions.

The present thesis delves into the potential of leveraging indirect encoding and evolvability during the neural network evolution process. To this end, we propose two algorithms, namely Quality Evolvability ES and Evolvability Map Elites, which enable direct selection for evolvability. We conduct an evaluation of the efficacy of these algorithms in the context of robotics locomotion tasks. Additionally, we investigate the necessary conditions for indirect encoding to be useful for learning and conduct experiments to verify our hypothesis in the domain of image recognition tasks.

Contents

Preface

List of Figures

List of Tables

Acknowledgements

I would like to express my deepest gratitude to my supervisors James Walker, and Daniel Franks, for their invaluable guidance, unwavering support, and constant encouragement throughout the entire process of this thesis. Their expertise, insightful feedback, and dedication have been instrumental in shaping my research and helping me grow as a scholar.

I am also profoundly grateful to my family and fiancée for their unwavering love, understanding, and continuous encouragement.

I would like to extend my heartfelt appreciation to my colleagues, especially my cohort of fellow IGGI students in York. I am grateful for the sense of community we have fostered, which has been an invaluable source of support and inspiration, both on a personal and professional level.

Lastly, I would like to acknowledge the great scientists whose work I have built upon and who have ignited my curiosity about evolvability. I vividly recall the profound amazement I experienced when I read "The Selfish Gene" for the first time. That initial sense of wonder has never wavered; if anything, it has only grown stronger with time.

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as references.

Some parts of this thesis have been published in conference proceedings; where items were published jointly with collaborators, the author of this thesis is responsible for the material presented here. For each published item the primary author is the first listed author.

Chapter 4 contains material from:

[5] Katona, Adam, Daniel W. Franks, and James Alfred Walker. "Quality Evolvability ES: Evolving Individuals With a Distribution of Well Performing and Diverse Offspring." In ALIFE 2022: The 2022 Conference on Artificial Life. MIT Press, 2021.

Chapter 6 contains material from:

[8] Katona, Adam, Nuno Lourenço, Penousal Machado, Daniel W. Franks, and James Alfred Walker. "Utilizing the Untapped Potential of Indirect Encoding for Neural Networks with Meta Learning." In Applications of Evolutionary Computation: 24th International Conference, EvoApplications 2021, Held as Part of EvoStar 2021, Virtual Event, April 7–9, 2021, Proceedings 24, pp. 537-551. Springer International Publishing, 2021.

Chapter 1

Introduction

1.1 Limitations of current AI

Scaling in both data and computation facilitated great progress in the field of Artificial Intelligence (AI). The last decade of AI development generated surprise after surprise with the capabilities achieved. Researchers solved many problems previously thought to be very complex with the help of deep neural networks such as playing the game GO [9] and Starcraft 2 [10] on superhuman level, solving the protein folding problem [11], generating amazing AI-assisted creative tools [12], and helping programmers write complex code, improving their productivity [13].

However, it is uncertain how far we can go by scaling up computation and data alone. Will the same approach of using gradient descent to train deep neural networks continue to work and produce ever more intelligent machines in the future? The three early pioneers of deep learning: Yoshua Bengio, Yann Lecun and Geoffrey Hinton argue that **scaling alone will not be enough to overcome the limitations of our current methods** [14], new breakthroughs will be necessary. One hint we can take to the source of such breakthroughs is to look at ourselves. We are the living proof that there exist a process which was demonstrated to be capable of producing human level general intelligence: Evolution.

1.2 The success of natural evolution

Even through we know evolution was capable of creating human level inteligence, we are still far away from replicating the success of evolution in silico. There are myriad details of the process of evolution which might be, or might not be important to replicating its success. On such detail is the evolution of evolvability.

Here we give a brief definition of what we mean by evolvability in this thesis. We define evolvability in general as the following: **The ability of an individual to produce offspring with good quality phenotypic variation.**

More specifically we define three kinds of evolvability based on what we consider good quality variation:

1. Adaptiveness of variation (how high the fitness of the offspring is?)
2. Innovativeness of variation (How different are the offspring from previously seen individuals?)
3. Diversity of variation (How different are the offspring compared to each other?)

We discuss why we choose this definition, how we can quantify evolvability and what other evolvability definitions exist in the next chapter.

1.3 Understanding evolvability with learning theory

Evolution of evolvability is an unintuitive concept. Even the question of whether evolvability evolves at all [15] is unclear, and there is still debate as to whether evolution of evolvability is caused by natural selection, or is a by-product of other evolutionary mechanisms [15, 16]. While this subject remains highly debated among biologists, researchers within the field of evolutionary computation are also increasingly captivated. Evolution of evolvability is desirable because it allows the possibility of accumulating the ability to evolve for a long time, potentially enabling the speed up of future evolution by many orders of magnitude, allowing us to utilize evolution in areas that were not practical before.

Recent work has enhanced our understanding by examining the evolution of evolvability from a learning theory perspective [17], leading to insights that are critical to consider when designing an evolution of evolvability algorithm. The reason it is possible for evolution to increase its ability to evolve in the future is similar to how it is possible for learning to generalize to unseen data [17]. It requires finding patterns in past experiences that generalize to future experiences. If some aspect of a genome was useful for generating adaptations in many different past environments, it might be useful in new, unseen environments as well. The job of an algorithm that aims to increase evolvability is to find these general patterns.

Looking at evolvability from a learning theory perspective, we identify three questions that need to be investigated before we are able to realize the vision of evolution of evolvability. These cover the three basic blocks of any learning process:

- data containing general patterns
- model with a capacity to learn these patterns
- an algorithm to drive this learning

In the next sections, we discuss what all of these components may look like for evolvability.

1.3.1 Evolvability Data

When we talk about evolvability data, we mean examples of evaluations of the ability of genomes to evolve further. We need to have many such examples to discover general patterns for which genes or combinations of genes tend to facilitate further evolution.

For evolvability patterns to be present in this data, it is essential that this data is diverse. This diversity has two important aspects:

- We need examples of many different genes and combinations of genes.
- We need examples of the same genes evaluated in multiple different environments.

If we only have examples from a single environment or task we have no way to guarantee that the genes which facilitate evolvability in this environment will also facilitate evolvability in different environments.

There are two different ways to acquire evolvability data from multiple environments. In our computer simulations, we can easily evaluate the same genes in many different environments simultaneously. We can even define a distribution of tasks, and evaluate the genes in this distribution, similarly to the meta learning problem formulation [18]. In nature, however, where there is some diversity present based on geographical locations, the vast majority of diversity comes from temporal differences in the environment. Many genes are old, and these genes have been tested through hundreds of millions of years through various environments as life on Earth continuously changes over time.

While nature is limited to temporal diversity due to constraints on reality, in our computer simulation we are free to simultaneously evaluate genes in many environments. Doing simultaneous evaluations has the benefit that we do not need to worry about forgetting. Nature has no way to go back to a past environment to check if a new gene which causes great evolvability now, would still be useful in past environments.

An open question is how do we provide endless new and interesting environments which will provide us with this kind of diverse evolvability data. Open-ended evolution is a field which attempts to answer this question[19]. There are several promising directions. The POET algorithm [20] aims to co-evolve a population of environments with agents. Another approach is to use a Large Language Model to evaluate the interestingness of tasks, and continuously create new learnable and interesting tasks [21].

1.3.2 Evolvability Model

The second question is how to provide evolution with a model which has sufficient capacity to learn evolvability. A powerful source of such capacity is the developmental program, which provides the instructions to translate a genotype into a phenotype. With a well-chosen developmental program, evolution is able to turn random genotypic variation into an advantageous distribution of phenotypic variation [17]. This is similar to how a Generative Adversarial Network [22] is able to generate a specific distribution from random

noise. A developmental encoding system is able to learn to produce a specific distribution of phenotypic variation from random mutations (as seen in Fig 1.1).

A simple example to demonstrate how such developmental decisions are able to affect evolvability is to imagine how the leg length of an animal is encoded [23]. By utilizing symmetry early in development, evolution can become more likely to explore the configurations where both legs are the same length and avoid exploring the probably poorly performing phenotypes with different left and right leg lengths. Different generative encodings can result in varying amounts of evolvability [24].

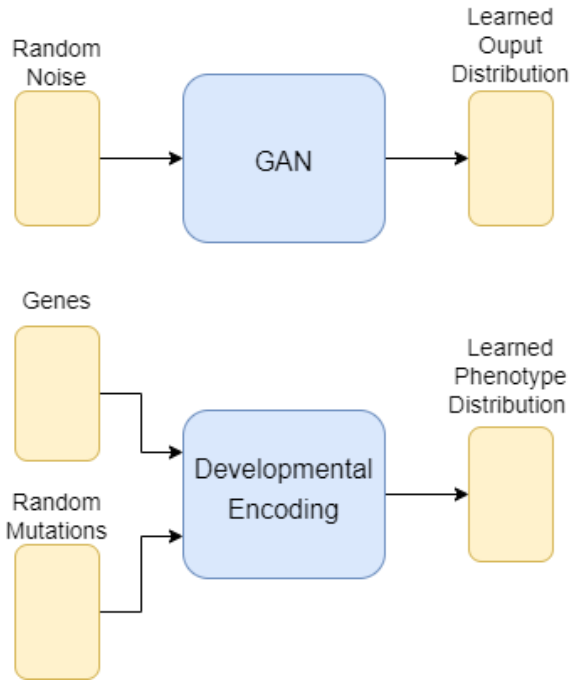


Figure 1.1: Developmental encoding can be used to learn evolvability, by learning a distribution of phenotypic variation. To elucidate this concept, we can draw parallels with the well-known example of Generative Adversarial Network (GAN). GANs learn to turn random noise into a distribution of output. Similarly, developmental encoding can learn to turn random mutations into a specific distribution of phenotypic variation.

Additionally, there is another source of learning capacity, which is simply selecting points in the genotype-space with good quality neighbourhoods. In the case of direct encoding, this is the only source, since there is no developmental program. The meta learning algorithm MAML [25] demonstrated the existence of such good quality neighbourhoods in the search space of large neural networks. Surprisingly, MAML is able to find points in the search space which are a few steps away from good solutions to many previously unseen problems.

1.3.3 Evolvability Algorithm

The third question is how to efficiently select for evolvability. It is not clear under which conditions evolvability can emerge in nature [15], whether it requires selection, or if it is a result of unsupervised learning [17]. In the case of evolutionary computation, we have the ability to directly select for evolvability. Although researchers have been talking about evolvability in evolutionary computation for many decades, the first algorithms that directly select from evolvability were introduced just a few years ago. (Evolvability ES [7], Evolvability Search [26]). The reason it took so long to realize algorithms that select for evolvability directly is it required improvements in both algorithms and computational capabilities. There are several existing algorithms that indirectly select for evolvability, and a few that directly select for it. We give an overview of these methods in chapter 2.

1.4 The main contributions:

We identified three necessary requirements for learning evolvability, these are

1. Evolvability data (past learning experience) which contain patterns which generalize for future learning
2. Evolvability model with a capacity to for learning evolvability (capacity to control what kind of phenotypic variation is generated as a result of random genotypic variation)
3. Evolvability algorithm, which optimizes for evolvability.

In this thesis we make contributions towards better understanding evolvability models and evolvability algorithms.

1.4.1 Contributions to evolvability algorithms

- We propose Resampling Expected Evolvability ES (REE-ES), a variant of the ES algorithm which is able to optimize for expected evolvability with linear scaling in

population size. This algorithm however has exponential scaling in problem dimension and thus can only be applied to very low dimensional problems ($d < 6$).

- We propose the approach of Quality Evolvability and the algorithm of Quality Evolvability ES. Quality Evolvability ES aims to find a single individual with a diverse and well-performing distribution of offspring. By doing so Quality Evolvability is forced to discover more evolvable representations.

- We propose Evolvability Map Elites, an algorithm with the goal of finding an archive of diverse, evolvable and fit individuals. This algorithm combines the benefits of quality diversity and quality evolvability, with the aim of utilizing the synergistic relationship between quality, diversity and evolvability.

1.4.2 Contributions to evolvability models

- We propose the hypothesis that we are unlikely to benefit from the powerful evolvability models which utilize indirect encoding using greedy learning algorithms. Greedy learning algorithms are algorithms which select for current performance and not for the potential for future performance. We provide evidence for this hypothesis by demonstrating that while direct encoding can outperform indirect encoding in an image classification task when using a greedy learning algorithm, indirect encoding performs better when using meta learning.

1.5 The main limitation

In this thesis, we study the three identified requirements mostly in isolation. In chapter 4 and chapter 5 we studied our new evolvability algorithms without using an indirect encoding to provide a large capacity for learning evolvability. In chapter 6. we studied indirect encoding but used a more established meta learning algorithm instead of a less well understood evolvability algorithm. The reasoning for this decision comes from the fact that studying things in isolation is easier, there are fewer variables to be concerned with. While this is true, some phenomena cannot be understood by looking at their components

in isolation [27]. In the future to deeply understand the requirements for evolvability to evolve, we need to study them in combination.

1.6 Thesis outline

The rest of the thesis has the following structure:

Chapter 2 is about evolvability. What is evolvability? How can we measure it? Can it evolve in nature? Can we evolve it in our simulations? We also discuss algorithms which directly or indirectly evolve evolvability and indirect encoding methods which provide a capacity to learn evolvability.

Chapter 3 is about related work and methods which this thesis builds on. This includes meta learning, neural networks, evolutionary algorithms, neuroevolution, and advances in computation.

In Chapter 4 we present two new evolvability algorithms. First is Resampling Expected Evolvability ES (REE-ES), which can optimize for expected evolvability with linear scaling in population size. Second, we introduce the concept of Quality Evolvability (QE), an approach which aims to find individuals who have good performing and diverse offspring. We present Quality Evolvability ES, a QE algorithm based on Natural Evolutionary Strategies. We demonstrate on robotic locomotion control tasks that Quality Evolvability ES, similarly to Quality Diversity methods, can learn faster than objective-based methods and can handle deceptive problems.

In Chapter 5 we introduce Evolvability Map Elites. A novel algorithm which aims to find a diverse set of highly evolvable and well performing individuals. We discuss the synergies and trade-offs between diversity, evolvability and performance. We introduced map elites variants with various evolvability pressures (elite selection, optimization for evolvability). We show that these changes result in some improvements in the objectives of map elites, however, these improvements are not consistent (not for all metrics / tasks) and not transformational.

In Chapter 6 we discuss the potential of indirect encoding. We propose the hypothesis that greedy learning is unlikely to utilize the capabilities of indirect encoding and

show with an image classification example that while indirect encoding performs worse in a greedy learning settings compared to direct encoding, the situation is reversed in the case of meta learning.

Chapter 7 is the conclusion, where we discuss future directions for evolvability research.

1.6.1 Source code

All the source code used in this thesis is available online.

- Chapter 4: `https://github.com/adam-katona/QualityEvolvabilityES`
- Chapter 5: `https://github.com/adam-katona/evolvability_map_elites`
- Chapter 6: `https://github.com/adam-katona/indirect_encoding_maml`

Chapter 2

Evolution of Evolvability

In this chapter, we discuss questions about evolvability: What is evolvability and how can we measure it? How can nature evolve evolvability? How can our evolutionary algorithms evolve evolvability? What kind of models are capable of learning evolvability?

2.1 What is evolvability

Different researchers use the term evolvability differently, a detailed review of the different evolvability concepts is given by Pigliucci [15]. In this section, we argue that some definitions are more useful for evolutionary computation than others.

2.1.1 Evolvability of the population

Some researchers consider evolvability to be a property of the population. Flatt et al.[28] defines evolvability as "the ability of a population to respond to selection", which mainly depends on the amount of standing genetic variation [15]. This is the evolvability definition selective breeders care about. The term "introducing new blood" refers to introducing alleles to the gene pool which were lost during the domestication process [29]. The population becomes more responsive to selection as we increase the standing genetic variation. While this kind of evolvability definition is useful for biologists and selective breeders, we argue that this is not as interesting for evolutionary computation for 2 reasons.

First, this definition ignores the quality of variation. In nature, variation is generated at a relatively slow rate and the genes responsible for it are continuously under selection. Most of the accumulated variation in the gene pool already has a long evolutionary history and proven viability. Variation in itself is not enough for evolvability, the quality of the variation is also important. This is especially obvious in the case of artificial evolution. We can generate arbitrary amounts and types of variation, but they will have very different effects on evolvability.

Second, standing variation can only explain short term evolvability. For evolvability to be sustained for a long time, a steady supply of variation needs to be generated. Differences in long-term evolvability therefore must depend on the ability to generate variation rather than the currently available variation in the population.

2.1.2 Evolvability of the individual

In the case of evolutionary computation, we care about evolvability definitions which are concerned with the ability of an individual to generate offspring with phenotypic variation. This kind of evolvability is about the potential to generate variation, whether it is realized or not.

Wagner and Altenberg [30][31] define evolvability as "the ability of the genetic operator/representation scheme to produce offspring that are fitter than their parents". Payne and Wagner [16] have a similar definition: "the ability of a biological system to produce phenotypic variation that is both heritable and adaptive". Wagner and Altenberg, further argues that "adaptation depends not only on how often offspring are fitter than their parents, but on how much fitter they are". The definition of Wagner and Altenberg can be summarized as the upper tail of the distribution of fitness effects. The distribution of fitness effects is a probability distribution of the change in fitness after applying the genetic operator once. It is important to note that this kind of evolvability is the property of the individual if we only consider mutations. If there is crossover the variation of the offspring of an individual also depends on the gene-pool of the potential sexual partners.

2.2 Quantifying evolvability

The above evolvability definitions emphasise the adaptiveness of the variation. The definition we use in this thesis however does not specify adaptiveness as the only important aspect of the variation. We define evolvability as **The ability of an individual to produce offspring with good quality phenotypic variation.**

We argue that while fitness is an important aspect of evolvability, there are other equally important aspects. We define three measures for the quality of variation. Each of these measures leads to a different and measurable evolvability definition. The decision about which definition to use should be based on which kind of evolvability is best suited for generalizing for future environments.

2.2.1 Adaptiveness

The first approach to quantify evolvability is to measure adaptiveness: which is a measure of both how often offspring have higher fitness than their parents, and the magnitude of the difference in their fitnesses.

Examples of algorithms which directly select for this kind of evolvability are ES[32], the meta learning algorithm MAML[25] and the evolutionary version of MAML, ES-MAML [33]. We will describe these algorithms later in this thesis.

2.2.2 Behavioural Diversity

The second approach to quantify evolvability is to measure the behavioural diversity of the offspring. Diversity can be measured by the variance or entropy of the distribution of behaviours. This approach requires a user defined behaviour characterization (BC) function, which turns some aspects of the behaviour of an individual into real-valued numbers. Examples of what kind of behaviour characterization functions were used in the past include: the final position of the robot [7], the ratio of time each leg of a robot is in contact with the ground [3] and the concatenated RAM state for each step for Atari games [34].

Care must be taken to define a BC where achieving diversity requires individuals to develop skills, so that evolution cannot simply exploit the BC function in a similar way to how it can exploit loopholes in not carefully designed fitness functions [35]. For example, in the case of a maze navigating robot, in a closed maze, the only way to reach new places is to learn to navigate and not run into walls. However, if the maze is open on one side, evolution can create individuals that wander outside of the maze, achieving high diversity of final positions without ever learning navigation skills [36]. Another related example of agents pursuing diversity without any benefit is the famous case of the reinforcement learning agent which got addicted to TV [37]. The agent was rewarded for being curious, which was measured by receiving observations that it was unable to predict. When the environment contained a TV screen with noise playing, the agent learned to stare at the screen instead of experimenting with the world and being curious that way.

Existing algorithms directly optimizing for this kind of evolvability are Evolvability Search [26] and Evolvability ES [7], which are described in more detail in the next section.

2.2.3 Innovation

The third approach is to define evolvability as the ability to generate innovation. Brookfield [38] defines evolvability as "the proportion of radically different designs created by mutation that are viable and fertile". However, innovation or "radically different" are subjective terms. Furthermore, something that was an innovation in the past, may no longer be considered innovative now. One way to measure innovation is to calculate novelty compared to an archive of previously observed behaviours. This is the approach used by Novelty Search [36]. It is important to note that while novelty can be used to quantify evolvability, Novelty Search does not directly select for evolvability. Evolvability search is interested in discovering individuals which are good at generating novelty, whereas Novelty Search aims to find novel individuals. The hope with this kind of evolvability definition is that if we can find the patterns which made an individual good at generating innovation in the past, maybe these patterns will be useful for generating innovation in the future. An algorithm that directly selects for this kind of evolvability is Novelty Search ES (NS-ES) [34].

2.3 Evolution of evolvability in nature

In this section we look at what are possible drivers of evolution of evolvability in nature, and what forces are at play. The reason it is difficult to explain evolution of evolvability is because evolution lacks foresight. Natural selection does not act on future potential, it selects based on current fitness alone. So how could evolution evolve evolvability? This is not well understood, but there are several theories which provide some insight.

2.3.1 Why study nature?

While it is interesting to study nature, if our goal is to replicate the success of natural evolution, we are not restricted by the same kind of constraints biology is. With our computers, we have complete freedom to design algorithms without consideration of biological feasibility. We can directly select for evolvability, or whatever we can calculate, we can divide individuals into groups, niches, and species however we see fit. Nevertheless studying natural evolution can still give us ideas and inspiration for building our artificial systems.

2.3.2 Evolvability as a side effect of adaptive mutations

One theory about how evolvability can increase over time is that changes in evolvability are side-effects of adaptive mutations. Because these changes are not driven by selection, there are examples of side effects which decrease evolvability as well as increase it.

Conrad [39] argues some changes that increase evolvability like some mutation buffering redundancies are in fact advantageous to the individual organism. "Some of the redundancies that confer stability on the phenotypic dynamics also serve to buffer the effect of genetic change." With redundancies, evolution can create adaptations freely without as much concern about the loss of some essential functionality. For evolutionary computation, this effect can be emphasised by introducing noise into our system. With more noise, the need for redundancy is increased.

Bloom et al. presents experimental evidence [40] for increased evolvability as a side

effect of adaptions which promote stability. A gene was manufactured which creates a more stable version of a protein. Then they created mutants of both the original and the more stable version of the gene. They found that there were multiple mutations which were adaptive for the more stable version, but were disruptive for the original version. They argue that most mutations are destabilizing whether they are beneficial or not. Stabilizing mutations increase evolvability, by increasing tolerance to random mutations. They further hypothesise, that innovative mutations, that radically change the behaviour of the protein are likely more destabilizing. If this is the case, the effect of robustness on evolvability should be even more significant. We have to be careful however about generalizing such results to non biological evolution. The mechanism which increased evolvability is very dependent on the specific domain, the stability of protein folding. In our simulated genetic encoding for various problems, robustness might not have such an intimate relationship with evolvability.

Clune, Mouret and Lipson [41] found that by adding a connection cost when evolving neural networks, the solutions become more modular, which in turn led to increased evolvability (solutions were found faster). The selection is driven not by selection for evolvability, but by selection for reduced resource use by having less connection in the brain. Evolvability was achieved as a side effect of this cost saving selection.

2.3.3 Evolvability through individual selection

Sometimes evolvability can be selected for, even if it results in decreased fitness. Conrad [39] explains that that evolvability facilitating adaptation can appear even if they are disadvantageous to the organism, because "Evolutionary advances are more likely to emanate from this portion of the population".

Reidl [42] discusses increasing evolvability through **removing risky independence**: "unnecessary, genetically redundant or risky independence or adaptive freedom of a single genetic unit is avoided by expedient dependency". Reidl argues that these changes can be enormously beneficial "Such adaptive advantages by systemization are so tremendous that the invention of a superimposed genetic unit must be expected, even if it would be a millionfold or trillionfold more unlikely than every other alteration within

the genome".

To understand how evolvability can be selected for by individual selection we have to think about what is fitness. People often talk about reproductive success, or the expected number of offspring of an individual as fitness [43]. A more helpful definition of fitness is a measure of the probability of survival. This kind of fitness definition does not apply to individuals, as they do not survive over evolutionary timescales, it applies to genes.

If we want to calculate the probability that a gene will still exist in a larger time frame, calculating fitness becomes recursive, we must consider the fitness of possible offspring through many generations. With this calculation in mind, it is easy to imagine an example, where a more evolvable (has a higher probability of generating adaptive offspring) gene can have higher fitness even if it decreases the likelihood to reproduce in the current generation.

This problem of explaining individual selection for evolvability can be approached somewhat similarly to the problem of explaining altruistic behaviour. Hamilton [44] introduced the term inclusive fitness after recognizing that a gene can also increase its survival by helping other individuals who share the same gene. There is a tradeoff between increasing the fitness of relatives and increasing the fitness of the individual. The same logic can be applied to evolvability. There is a trade-off between increasing the individual's own probability to reproduce with its offspring's probability to reproduce through many generations. If a gene is less adapted to its environment but a champion evolver, it might have a better chance of surviving in the long term.

There are examples where selection decreases evolvability. Clune et al. [45] demonstrates that evolution fails to optimize mutation rates on rugged fitness landscapes. Clune et al. found that evolution will decrease the mutation rate towards zero once it reached a local peak, if the distance to the next peak is large enough. This can be explained when we consider the recursive fitness calculation. When the peaks are far away, the loss in expected fitness in the next generation is not worth the gain of low probability of finding the next peak. Here the trade-off is reversed, recursive fitness is larger when evolvability is decreased.

A similar phenomenon is shown by Wilke et al. [46], by demonstrating that evolution

prefers the flatter lower fitness peak, to the narrow high fitness peak. This is because recursive fitness is higher in the flatter peak.

It is important to notice that the trade-off between short term fitness and long term evolvability depends on how large the selection pressure is. If the selection is weaker, evolvability will be a larger factor, since the same amount of direct fitness advantage results in less advantage in the number of expected children. Lynch [47] argues that lower selection leads to increased complexity: "substantial evidence exists that a reduction in the efficiency of selection drives the evolution of genomic complexity".

When evolving the mutation rate in evolutionary computation, it is suggested [48] to first mutate the mutation rate, and use this new mutation rate to evolve the rest of the genome. This way selection has a little more information to evaluate the appropriateness of the mutation rate. By using this trick, we can bias the recursive fitness towards long term survivability instead of direct fitness (the probability to reproduce in this generation)

2.3.4 Evolvability through higher level selection

The problem with evolving evolvability through purely individual selection is that the effect is relatively weak. The trade-off between direct fitness and evolvability allows only a small room of parameter range where evolvability has an advantage. Zhong and Priest[49] makes this point the following way: "Because natural selection lacks foresight and tends to fix alleles that maximise current fitness regardless of the consequences for future evolutionary potential of the population, evolvability is generally not expected to be selected at the level of individuals".

Dawkins [50] argues that some embryologies are eager or reluctant to evolve in a certain direction, and this property can be selected by some kind of higher level selection. Dawkins uses the example of body segmentation as a big step in evolvability. The first animal with a segmented body was probably a freak, it had two bodies instead of one, but it somehow survived and reproduced and opened the gate for evolving into all kinds of specializations.

Lynch [47] argues for the weakness of higher level selection the following way: "This [higher level selection] requires a significantly subdivided population structure, with levels

of evolvability being positively correlated with population longevity and/or productivity" "Because populations survive longer than individuals, such group selection is expected to be a much weaker force than individual selection, and necessarily operates on much longer time scales."

An example of higher level selection was shown by Goldberg et al. [51] by observing the nightshade plant family (Solanaceae). Some of these plants can self-fertilize, which provides a short term advantage compared to plants that cannot. The plants without self-fertilization are found to diversify at significantly higher rates. This results in a strong species selection against self-fertilization.

2.4 Evolvability algorithms

In the previous section, we discussed what might drive evolvability in nature. In the case of computers, however, we are not limited to methods which are biologically feasible. In this section we discuss the various methods researchers came up with which either directly or indirectly select for evolvability.

2.4.1 Indirect Selection

There are various mechanisms that produce indirect selection for evolvability [26]. One mechanism is to introduce regular mass extinction events [52], freeing up many niches. Evolvability is indirectly rewarded because lineages that have the ability to radiate to the empty niches faster have a higher chance of surviving the next extinction event. A similar situation can be achieved with constant goal switching [53], which indirectly rewards individuals that developed the ability to adapt between goals faster. Novelty search is another method that rewards radiating into new niches and therefore indirectly rewards evolvability [54]. Another mechanism is innovation protection, which protects new genes for a few generations. This protection allows genes that increase evolvability to stay alive long enough to realize the benefits of increased evolvability [1, 55].

These are all indirect methods because they do not directly reward evolvability, but create a situation where lineages that are evolvable have some benefit. These methods

are vulnerable to being exploited. For example in the case of novelty search, the indirect selection for evolvability can be exploited by individuals who happen to be novel without being evolvable.

2.4.2 Direct Selection

In this section we dicuss the few algorithms which directly select for evolvability, instead of indirectly selecting for some kind of proxy of evolvability. These algorithms are not exploitable by individuals which are not evolvable, but seem evolvable because of their proxy metric. Even though techniques to increase evolvability have been discussed for several decades, the first technique to directly select for evolvability, Evolvability Search [26], was only reported recently. Evolvability Search quantifies evolvability by calculating the behavioural diversity of offspring. This is achieved by sampling and evaluating the offspring of every individual in the population, only to discard all these evaluations after evolvability is calculated. Discarding all these evaluations makes this technique extremely computationally expensive.

Another similarly expensive technique is ES-MAML [33], the evolutionary version of the meta learning technique MAML. ES-MAML selects for the adaptiveness aspect of evolvability. The adaptiveness can be calculated by applying various adaptation operators. For example, the adaptation operator can be another ES update, which includes evaluating and subsequently discarding the offspring of each individual in the population, making ES-MAML as expensive as evolvability search.

Finally, there is a family of algorithms which is able to select various aspects of evolvability without this expensive computational cost. These are the Evolutionary Strategies ES [32] algorithms which is a special case of Natural Evolutionary Strategies [56]. Here we are talking about the recently published gradient-based ES algorithm, not the old evolutionary algorithm with the same name, for example, $1+1$ ES. Normal ES selects for the adaptiveness aspect of evolvability, Evolvability ES (E-ES) [7] selects for the behavioural diversity aspect while Novelty Search ES (NS-ES)[57] selects for innovativeness aspect. In this thesis we use these algorithms in both chapter 4 and chapter 5. We give a detailed discussion of these algorithms and how they can select for evolvability without the large

computational cost of the other approaches in chapter 3.

2.5 Evolvability model

We claimed earlier that evolvability is achieved by evolution having control over the kind of phenotypic variation it can generate. In this section, we discuss the main mechanism that makes this control possible: indirect encoding.

2.5.1 The Potential of Indirect Encoding

In this work, we argue that indirect encoding is worthy of our attention because it has two interesting properties, which direct encoding lacks:

1. Indirect encoding can control the exploration during learning by making changes in promising directions more sensitive and changes in less promising directions insensitive.

2. Indirect encoding can reuse parameters multiple times, making it possible to learn regular structures.

Controlling Exploration

Indirect encoding is capable of controlling exploration during learning by modifying the type of variation mutations can cause. This is possible since the genotype-phenotype map has the ability to transform random genotypic variation to an advantageous distribution of phenotypic variation [17].

A simple example which is often used to demonstrate this property is how nature encodes development plans for symmetric bodies [23, 26]. Because of the way the developmental program for the body is encoded, it is easier for evolution to change the length of both limbs together, than to change them separately, which is probably a useful way to explore possible space of body configurations.

A similar concept that describes the same phenomena is developmental canalization. Indirect encoding has the ability to entrench certain phenotypic features, making them difficult to change, the same way water can dig a canal, making the path of future flow more stable. A great example of how canalization can emerge in artificial evolution is given in [23]. In their experiments, developmental canalization made certain good quality variation more common, increasing the ability to innovate.

One of the most successful algorithms that can control its own exploration is Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [58]. CMA-ES uses an exploration strategy of adapting the covariance matrix of a multivariate Gaussian distribution in a way that more promising directions are explored more. By using indirect encoding we can allow the algorithm to automatically discover exploration strategies instead of manually inventing and incorporating them into our algorithms.

Reusing Parameters

Another important property of indirect encoding is the ability to reuse parameters multiple times. We argue that this is actually a similar property to controlling exploration since having many separate parameters that always change together or having one parameter which is reused many times achieves the same results.

Indirect encoding allows the reuse of information to build regular and modular structures [59, 60]. We already know how beneficial some structures are, for example, the convolution is reusing the convolutional kernel weights many times at different locations in the image, which makes learning vision tasks efficient. By using indirect encoding we open the possibility to discover such useful structures automatically, even when using a fully connected architecture [61].

The Vision of Indirect Encoding

By using an indirect encoding, we can allow our algorithms to automatically discover different modular architectures, and exploration strategies, instead of manually inventing and coding them. The vision of automatically discovering these kinds of representations with the ability to learn effectively for many kinds of problems is extremely alluring [60].

Richard Sutton's bitter lesson argument [62] postulates that general methods which scale well with computation eventually always outperform methods for which the AI researchers build in extra handcrafted knowledge. For this reason, researchers should concentrate their main efforts on studying general and scalable methods. We hypothesise that utilising indirect encoding will allow us to discover efficient learning systems automatically in a data-driven way in the spirit of Sutton's argument.

2.5.2 Definition of indirect encoding

What is indirect encoding? From the name indirect encoding alone, one could naively define it as having some transformation between the genotype (the representation we store individuals and apply genetic operators to), and the phenotype (a representation which is evaluated in an environment).

However, this naive definition does not capture the essence of why indirect encoding is so powerful. In the case of evolvability, we are interested in indirect encoding because it is the source of the capacity of the model to learn to be more evolvable. We will discuss why this definition does not tell us much about the capacity for evolvability, and we propose a definition that is more interesting for evolvability. First, let us discuss a few examples of encodings:

Direct representation

An example of direct representation is a directly encoded fixed topology neural network. In this example, each weight and bias of that network is represented as a separate parameter. Most deep learning research uses this kind of direct representation.

Simple transformation

An example of a simple transformation is encoding a 2-dimensional point with polar coordinates. In this case, the genotype is represented with a radius r and an angle ϕ. A simple transformation is used to get the phenotype, (the Euclidean representation) of

Example	Naive Definition	Useful for evolvability
Direct encoding	NO	NO
Simple transformation	YES	NO
Explicit complex transformation	YES	YES
Implicit complex transformation	NO	YES

Table 2.1: Summary of which example is considered indirect encoding according to the naive definition, and which one is useful for evolvability

the points: $x = r * cos(\phi)$, $y = r * sin(\phi)$. This simple transformation represents a one-to-one mapping, each location in the genotype space corresponds to a single, unique location in the phenotype space and vice versa.

Explicit complex transformation

An example of an explicit complex transformation is HyperNEAT [63]. In the case of HyperNEAT, the genotype is the parameters of a relatively small function. This function then can be used to generate the weight of a large network (the phenotype). This allows HyperNEAT to encode a large network with a small number of parameters. We call this example explicit because the transformation is defined by the experimenter.

Implicit complex transformation

In the previous example, the experimenter defined the transformation between genotype and phenotype. However, it is possible for this transformation to emerge on its own. Let us imagine a task of evolving programs to solve some kind of task. The genotype is binary code, which is executed on a computer. We can imagine that evolution evolves a compiler as the first part of its genotype, which when run, compiles the second part of the genotype to different instructions. In this case, the second part of the genotype has now a different meaning, it represents the code in the language defined by the first part of the genotype. This is an example where the experimenter did not define explicitly the transformation, but it emerged automatically.

Indirect encoding discussion

According to the simple definition, the simple transformation and the explicit complex transformation are considered to be indirect encoding. This is because there is a clear distinction between genotype and phenotype with an explicit mapping between them. But are these examples useful for evolvability?

In the previous section, we discussed that evolvability can only be evolved if the model has a capacity for evolvability. This means the model must have the ability to change the kind of phenotypic variation it is able to generate via random mutations. The model must be able to make changes in some directions in phenotype space more likely, others less likely. Can evolution do that in our examples?

In the case of simple transformation, the answer is no. While it is true that the kind of variation generated in polar coordinates is different compared to the ones in Euclidean coordinates, evolution has no control over it. This is because the mapping between the two spaces is strictly one to one. In the case of the explicit complex transformation example, there are many genotypes to generate the same phenotype with HyperNEAT. What is different between these genotypes is what kind of variation they generate in phenotype space when perturbed. Evolution can choose between all these genotypes based on which generates the best quality variation. The same case is true for the implicit complex transformation example. Evolution has control over the type of compiler it creates, allowing it to generate the same program in all kinds of languages. It also has a many to one encoding. Furthermore, some of these languages will be much easier to evolve than others.

What matters for evolvability is whether the encoding allows for evolution to control exploration, which is only possible if the mapping between genotype space and phenotype space is many to one. It is important to note that **being many to one is a necessary but not sufficient condition** for an encoding to provide control for evolvability. For example, if we encode a number as a sum of two other numbers. $x = a + b$. This is a many to one encoding, there are many ways to choose the a and b which result in the same x. But such an encoding does not allow evolution to change evolvability. When a mutation is applied to the a and b, the resulting distribution of x will be the sum of the mutations and does not depend on a and b. The transformation has to be such that the resulting

phenotype distribution depends on the encoding parameters.

This leads us to the definition of indirect encoding we use in this thesis. An encoding is indirect if it satisfies both of these conditions:

- It defines a many to one transformation. (either explicitly or implicitly)
- Different genotypes representing the same phenotype have the possibility to produce different distributions of phenotypic variation when perturbed. (the transformation is complex)

Chapter 3

Related work

In this chapter, we go over some other algorithms and techniques which are related to the work presented in this thesis.

3.1 Meta learning

In the previous chapter we discussed that in the case of evolvability, the goal is to learn representations which produce good quality phenotypic variation when random variation is applied to the genotype. Evolvability is a special kind of learning to learn, or meta learning algorithm. Meta learning aims to learn something (meta-knowledge) about the learning process, to speed up future learning. There are all kinds of components of the learning process besides evolvability that can be learned. One such component is the initial weights of the model. It is possible to find weights which allow fast adaptation for a distribution of tasks. This is what is learned by several methods like MAML [25] or Reptile [64]. Meta-SGD [65] and MAML++ [66] also learn initial weights, but they additionally learn per layer per step adaptation learning rates. Evolved policy gradients [67] learns a reward function, which when optimized allows for fast learning for reinforcement problems. DARTS [68] learns network architectures which adapt fast. Meta-LSTM [69] learns optimizers which allow fast learning. In the next section, we go over the MAML algorithm in more detail, as this is the meta learning approach we use in chapter 6.

3.1.1 MAML

The goal of MAML is to find initial parameters θ, that allow for fast adaptation for many different tasks, by only taking a single gradient step. For meta learning, the problem is formulated as a set of tasks. For each training step we sample this task set, and divide the data for each task into a training set and a testing set. There are 2 kinds of updates. The fine-tuning update is defined as a normal gradient update. For each task, we adapt the parameters by minimizing the loss function using the training sets of these tasks, as seen in equation 3.1. $\mathcal{D}_i^{tr}$ refres to the training set of the i-th task, $\mathcal{L}$ is the loss function, α is the learning rate. θ' is the finetuned parameters. The second kind of update is the meta update. (see equation 3.2). The meta loss is defined as the loss of the finetuned parameters θ' on the test set of the tasks $\mathcal{D}_i^{ts}$. Calculating the meta gradient requires us to differentiate through a gradient step (see equation 3.2), which means we also need to calculate second-order gradients. It is also possible to ignore the second-order gradients because of their high computational cost, which leads us to first-order MAML.

$$\theta' = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i^{tr}) \tag{3.1}$$

$$\mathcal{L}_{meta} = \sum_{task\, i} \mathcal{L}(\theta', \mathcal{D}_i^{ts}) = \sum_{task\, i} \mathcal{L}(\theta - \alpha \nabla_\theta L(\theta, \mathcal{D}_i^{tr}), \mathcal{D}_i^{ts}) \tag{3.2}$$

3.2 Evolutionary Strategies

One of the core algorithms used in this thesis is Evolutionary Strategies (ES). ES is a special case of the more general Natural Evolution Strategies (NES) [56] algorithm, which aims to calculate the gradient of the expected fitness of a parameterized search distribution p_ϕ with respect to the distribution parameters ϕ. In the case of ES, this is an isotropic normal distribution, parameterized by a center individual: $p_\phi = \mathcal{N}(\phi, \sigma)$ where $\sigma \in \mathbb{R}$. The cost function is defined as an expectation over the distribution $J = \mathbb{E}_{\theta \sim p(\phi)} F(\theta)$. The update rule is derived by applying the "log-likelihood trick" [56] so the gradient of the expectation can be expressed with the expectation of a gradient (eq.3.3), which can be

approximated with samples.

$$\nabla_\phi \mathbb{E}_{\theta \sim p_\phi} F(\theta) = \mathbb{E}_{\theta \sim p_\phi} \{F(\theta) \nabla_\phi log p_\phi(\theta)\} \tag{3.3}$$

Algorithm 1: ES

Input: Noise standard deviation σ, initial policy parameters θ_0, population size n, gradient optimizer, fitness function F

for *t = 0, 1, 2, . . .* **do**
- Sample $\epsilon_1 ... \epsilon_n \sim \mathcal{N}(0, I)$
- $F_i = F(\theta_t + \sigma \epsilon_i)$ for $i = 1, ..., n$
- $r =$ fitness_shaping(F)
- $grad = \frac{1}{n\sigma} \sum_{i=1}^{n} r_i \epsilon_i$
- $\theta_{t+1} = optimizer(\theta_t, grad)$

end

Previous work found that using a rank-based fitness shaping tends to improve performance by reducing the effect of outliers in the population [32]. We included this step as well in the description of the algorithms with the optional function fitness_shaping.

ES has several differences compared to traditional finite difference approximators. The effect of optimizing for the expectation over a normal distribution can be imagined as using normal gradient descent on a blurred version of the fitness landscape [32]. Another consequence of optimizing for expected fitness instead of the fitness, is that ES aims to find solutions that are robust to perturbations [70].

3.2.1 Variance and entropy as evolvability

To create an evolvability variant of ES, there is a need for an exact way to quantify evolvability. In this case, we are talking about the behavioural diversity aspect of evolvability (as described in chapter 2). This means we want to quantify the diversity of the offspring distribution. Gajewski et al. [7] introduced two ways to do this.

The first one is to use the variance of the distribution (Eq. 3.4.). Here k is the number of dimensions in the behaviour characterization function BC, and $\overline{BC}$ is the mean of the behaviour for the sample.

$$Evolvability(\phi) = \sum_{j=0}^{k} \mathbb{E}_{\theta \sim p_\phi} (BC_j(\theta) - \overline{BC}_j)^2 \tag{3.4}$$

The second one is to use the entropy of the distribution (Eq. 3.5.). This one is a little more complicated since we need to calculate the probability of each behaviour via kernel density estimation.

$$Evolvability(\phi) = -\mathbb{E}_{\theta \sim p_\phi}(-log(p(BC(\theta)))) \tag{3.5}$$

3.2.2 Expected Evolvability ES

The simplest way to create an ES variant that directly selects for evolvability is to replace fitness with evolvability in the cost function: $J = \mathbb{E}_{\theta \sim p(\phi)} Evolvability(\theta)$. Let us call this variant Expected Evolvability ES (EE-ES), to differentiate it from Evolvability ES [7] which we will discuss in the next section. The problem with Expected Evolvability ES is that it requires evaluating the evolvability of each individual in the population. Since evaluating evolvability requires sampling the offspring of that individual, EE-ES requires $O(n^2)$ (where n is the population size or sample size) evaluations instead of the $O(n)$ of normal ES. For example with a population size of a thousand, while ES requires a thousand evaluations per iteration, EE-ES requires a million. This makes EE-ES prohibitively expensive for most interesting problems.

3.2.3 Evolvability ES

Gajewski et al. [7] recognized that there is a different way of creating an ES algorithm that directly selects for evolvability. Evolvability is itself defined as an expectation (both the variance or entropy definitions are expectations, as can be seen in Eq. 3.4 and Eq. 3.5), so the log-likelihood trick which is used to derive the ES update rule can be applied directly to evolvability, instead of the expected evolvability. The resulting algorithm, Evolvability ES, has the cost function: $J = Evolvability(\phi)$, compared to the cost function of EE-ES $J = \mathbb{E}_{\theta \sim p(\phi)} Evolvability(\theta)$. Evolvability ES no longer needs to calculate the evolvability of every individual, making Evolvability ES as fast as normal fitness-based ES. It is important to note that the new algorithm no longer has the blurring and robustness-seeking property of ES, and is more like a traditional finite difference approximator.

Algorithm 2: Evolvability ES

Input: Noise standard deviation σ, initial policy parameters θ_0, population size n, gradient optimizer, behaviour characterization BC

for *t = 0, 1, 2, . . .* **do**

Sample $\epsilon_1 ... \epsilon_n \sim \mathcal{N}(0, I)$

$EVO_i = (BC(\theta_t + \sigma\epsilon_i) - BC_{mean})^2$ for $i = 1, ..., n$

$r = \text{fitness\_shaping}(EVO)$

$grad = \frac{1}{n\sigma} \sum_{i=1}^{n} r_i \epsilon_i$

$\theta_{t+1} = optimizer(\theta_t, grad)$

end

3.3 Neural Networks

In this thesis, in all of our experiments, we are evolving neural networks. There are several interesting properties which make neural networks attractive.

3.3.1 Universal approximator

Neural networks have the favourable property that they are universal function approximators [71], meaning they can approximate any function with arbitrary precision. This property can be illustrated for any one dimensional function. Consider two neurons, one acts as step function, the other acts as an inverse step function. The sum of these neurons will be a bar with the width equal to the difference between the threshold of the two neurons. Then we can combine many of these bars to approximate any function. This kind of visual proof can also be extended to higher dimensions, with an n-dimensional bar requiring n pairs of neurons. While this property is interesting, a single hidden layer with enough neurons can approximate any function, however, in practice, deep networks work much better.

3.3.2 Deep networks

If a neural network has several hidden layers it is called a deep network. Deep networks can be more expressive than shallow networks because they can compose the functions of previous layers in exponentially many ways [72].

In case the network has a piece-wise linear activation function (for example rectified

linear function) the expressivity of the network can be measured by the number of different linear segments of the function represented by the neural network. The exponential relationship between the depth of the network and expressivity was demonstrated by [72], by showing how each layer can be used to apply a mirroring or folding to the input space. Each of these foldings can potentially double the expressivity of the network.

3.3.3 Vanishing gradient

Neural networks are usually trained with gradient descent optimization. In the case of deep networks, gradient descent can become problematic, since during backpropagation the gradient is multiplied with the derivative of the activation function many times. This can lead to the vanishing/exploding gradient problem [73]. Recurrent networks especially suffer from the vanishing/exploding gradient problem. This is because when we do the backpropagation through time, we must multiply the gradient with the derivative of the activation function every timestep we consider. There are several techniques to mitigate this problem, for example, the residual layer [73] or in the case of recurrent networks the long short term memory layer or LSTM. Newer architectures like the transformer [74] also use residual connections.

3.4 Neuroevolution

When neural networks are trained with evolutionary algorithms, we are talking about neuroevolution. While currently, gradient descent training works much better compared to evolution, there are several interesting properties of evolution which make neuroevolution an interesting research area.

3.4.1 Non differentiable aspects

While gradient descent optimization currently outperforms evolutionary algorithms in optimizing neural networks, it requires the model to be differentiable. Evolutionary computation has no such requirements, which makes it possible to optimize the topology of

the network, learning rules, the amount of plasticity in different areas, and other non differentiable aspects we have not thought of as yet. [60]

3.4.2 Vanishing gradient in neuroevolution

In the case of neuroevolution, we might first think that the vanishing gradient problem is not an issue, because evolution does not require gradients. However this is not the case, the gradient indicates the sensitivity of a parameter, and if a parameter is too sensitive or insensitive, the mutations will cause too much or too little perturbation to be effective. Lehman [75] showed that using the gradients of the network output to a reference input set can be used to inform the mutation operator to take sensitivity into account, which can lead to better performance.

3.4.3 Competing convention problem

A challenging aspect of neuroevolution is the competing convention problem [76]. A neural network represents information in the activation of its neurons. The order of these neurons however is arbitrary, the network can represent the same information even if we permute the order of the neurons. Evolution is a population based algorithm, and if different individuals in the population learn different conventions to represent information, crossing them over is likely to be destructive.

There are two approaches to dealing with the competing convention problem. Some researchers just stopped using crossover [32] [77] and only use mutations. Mutation only neuroevolution turns out to work surprisingly well on large networks (millions of parameters) reaching comparable performance to the 2013 breakthrough results on atari games using a deep Q network [78].

Another approach was introduced by Stanley [1] by using historical markings. Historical markings allow the imitation of two natural processes, speciation and synapsis. If two populations become too different from each other, reproduction is not possible any more, the original species become two species, and the population has speciated. Synapsis is the process of aligning homologous genes, genes which express the same trait [79].

Synapsis helps prevent crossing over incompatible genes (genes with competing conventions).

Historical markings allow the experimenter to line up genes which share a common history, emulating synapsis. The same markings can also be used to easily measure the distance between individuals, which allows to cluster them into species. This approach was later applied to neuroevolution with indirect encoding [63], [80] and gene regulatory networks [81] to facilitate crossover in these domains.

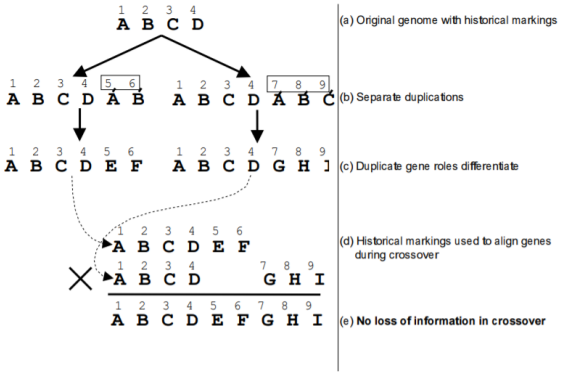


Figure 3.1: Historical markings [1] allow crossover to be effective by overcoming the competing convention problem

3.5 XLA

There has been huge progress in not just the hardware, but the software which allows evolution to run on the scales which make it competitive with some basic deep learning methods. One such advancement is the XLA compiler by Google [82]. XLA compiles linear algebra graphs into accelerator (GPU or other) code. Before XLA, users who wanted to run their linear algebra code on GPUs had 2 choices. One option was to use an established tensor library like tensorflow or pytorch. This allowed to write very flexible and simple code to express complex linear algebra computations, with automatic differentiations. However, these libraries had a significant performance penalty, the main source is which is the use of precompiled kernels for each operation. Each operation needed a separate kernel launch, and the result needed to be copied back to the device memory. Because of this, the user misses out on potential optimizations by combining operations which would keep the intermediate result in registers, saving the copy back to memory.

The second option was to write custom CUDA kernels, which are fast and energy efficient, but error prone, requiring a lot of expertise and development time.

With XLA, it becomes easy to run all kinds of applications on the GPU that use linear algebra and the operations are sufficiently large for efficient parallelization, such as physics engines or various evolutionary algorithms. The JAX [83] library offers a numpy [84] like API for generating just-in-time compiled XLA programs.

In chapter 5 we use the XLA-based Brax [85] physics engine to run thousands of simulations on the GPU concurrently. A few years ago before this technology existed, we would have required thousands of cpu-s to achieve similar performance. XLA helped radically reduce the cost and energy consumption of such experiments, making them accessible to individual researchers.

Chapter 4

Quality Evolvability: Evolving Individuals With a Distribution of Well Performing and Diverse Offspring

4.1 Introduction

One of the most important lessons from the success of deep learning is that learned representations tend to perform much better at any task compared to representations we design by hand. Yet evolution of evolvability algorithms, which aim to automatically learn good genetic representations, have received relatively little attention, perhaps because of the large amount of computational power they require. The recent method Evolvability ES [7] allows direct selection for evolvability with little computation. However, it can only be used to solve problems where evolvability and task performance are aligned. In this chapter, we propose Quality Evolvability ES, a method that simultaneously optimizes for task performance and evolvability without this restriction. This is achieved by using nondominated sorting inside the ES update. Our proposed approach Quality Evolvability has a similar motivation to Quality Diversity algorithms but with some important differences. While Quality Diversity aims to find an archive of diverse and well-performing, but potentially

genetically distant individuals, Quality Evolvability aims to find a single individual with a diverse and well-performing distribution of offspring. By doing so Quality Evolvability is forced to discover more evolvable representations. We demonstrate on robotic locomotion control tasks that Quality Evolvability ES, similarly to Quality Diversity methods, can learn faster than objective-based methods and can handle deceptive problems.

4.1.1 Chapter contributions

This chapter makes the following contributions:

- We propose Resampling Expected Evolvability ES (REE-ES). REE-ES optimizes for expected evolvability with linear scaling in population size, unlike the naive Expected Evolvability ES which has quadratic scaling. REE-ES however have exponential scaling in problem dimension and thus can only be applied to very low dimensional problems ($d < 6$)
- We propose Quality Evolvability, an approach that directly selects for evolvability by finding individuals with a diverse and well-performing distribution of offspring, and we present Quality Evolvability ES a method based on Evolvability ES, which is able to select for Quality Evolvability with a low computational cost.
- We demonstrate on robotic locomotion tasks, that Quality Evolvability ES is able to outperform the objective based variant of the algorithm, can be applied to problems where evolvability and performance are not aligned, and is able to handle deceptive problems.

The main difference between our work, and previous work [4, 86] which uses non-dominated sorting to simultaneously select for diversity and fitness is that our work selects for the expected diversity of the offspring, compared to the diversity of the population. This is possible to do efficiently because of the special properties of ES to estimate gradients of expectations. Our source code is available at: *https://github.com/adam-katona/QualityEvolvabilityES*.

4.2 Resampling Expected Evolvability ES

In the previous chapter (3), we talked about the most straightforward way of creating a variant of ES which selects for evolvability: just replacing fitness with evolvability in the ES objective function. This leads to Expected Evolvability ES (EE-ES) which has the objective function of $J = \mathbb{E}_{\theta \in p(\phi))} Evolvability(\theta)$ instead of the objective function of normal ES: $J = \mathbb{E}_{\theta \in p(\phi))} F(\theta)$. We also discussed the problem with EE-ES, that it has $O(n^2)$ complexity in population size.

In this section, we introduce a variant of the algorithm called Resampling Expected Evolvability Evolutionary Strategies (REE-ES) which is able to estimate the evolvability of every individual of the population while keeping linear scaling with population size. We also discuss the serious limitation of our new algorithm variant. We show that our new algorithm while having linear scaling in population size, has exponential scaling in problem dimension, making it only practical for very small dimensional problems ($d < 6$).

4.2.1 The complexity problem of EE-ES

The problem with Expected Evolvability ES is that it requires evaluating the evolvability of each individual in the population. Since evaluating evolvability requires sampling the offspring of that individual, EE-ES requires $O(n^2)$ (where n is the population size) evaluations instead of the $O(n)$ of normal ES. For example with a population size of a thousand, while ES requires a thousand evaluations per iteration, EE-ES requires a million. This makes EE-ES prohibitively expensive for most interesting problems.

4.2.2 Motivation

A legitimate question to ask is why we need a new Evolvability ES variant with linear scaling in population size when we already have Evolvability ES [7] which already achieves this. Evolvability ES is not a traditional ES algorithm, it estimates the gradient of Evolvability of the central parameters, instead of the expected evolvability of the population. It can do this because evolvability itself is defined as an expectation, so using the ES update formula can be directly derived for evolvability. This makes Evolvability ES more

similar to a finite difference gradient estimator than to an evolutionary strategies algorithm. This is important because there are several properties of ES which are lost for Evolvability ES. For example, ES can be imagined as acting like doing gradient descent on the Gaussian blurred objective function [32]. Also, ES seeks solutions that are robust to perturbations[70]. These properties make an ES version which optimizes for expected evolvability instead of evolvability desirable.

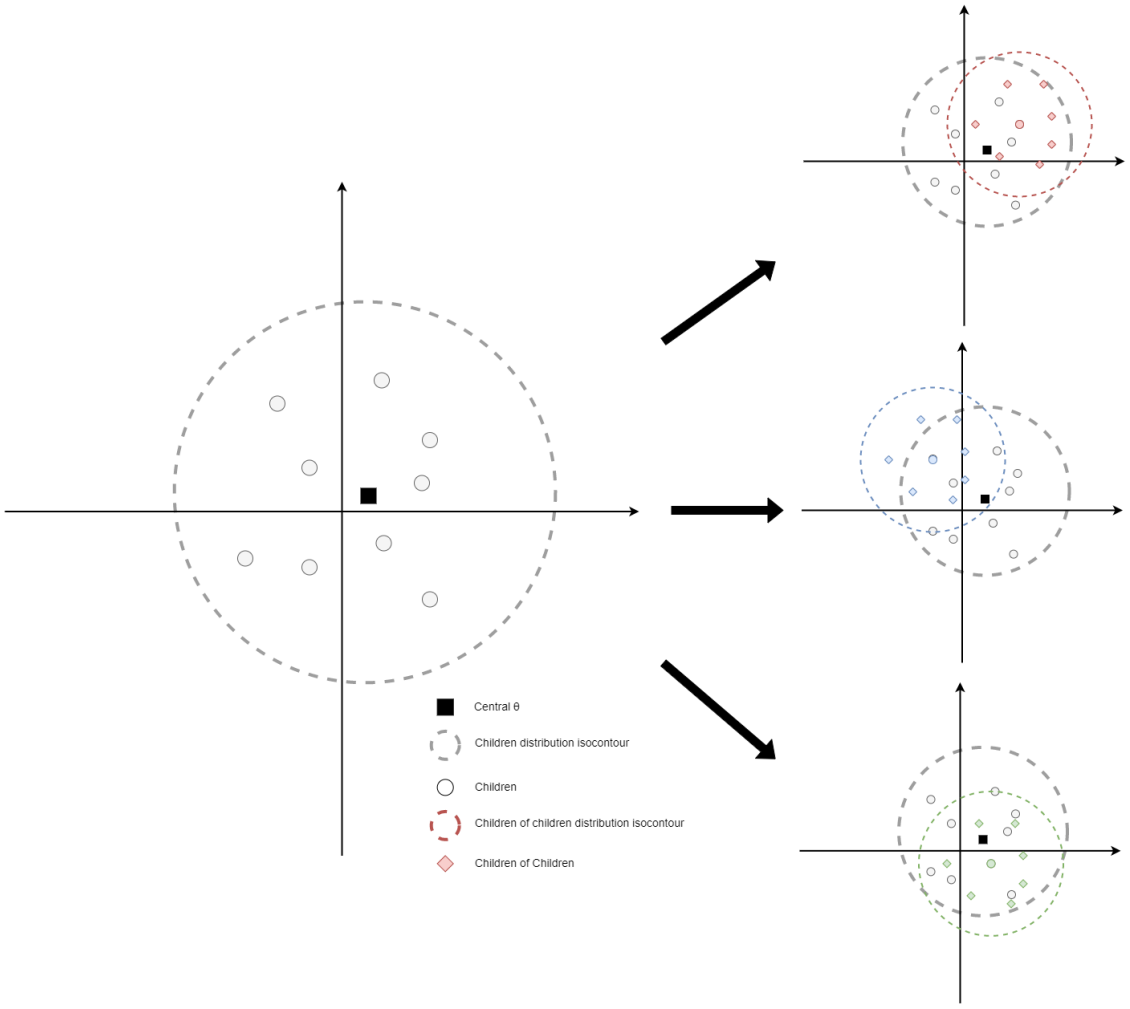


Figure 4.1: Figure to illustrate the evaluations needed for calculating the expected evolvability for EE-ES. For normal ES (on the left) we sample the n individuals from the distribution $\mathcal{N}(\theta, \sigma)$, where n is the population size. For EE-ES (on the right), we need to evaluate evolvability for every individual in the population, which means we need to sample n grandchildren from each child θ_i, with $\mathcal{N}(\theta_i, \sigma)$, requiring a total of n^2 evaluations.

4.2.3 REE-ES

The reason $O(n^2)$ evaluations are necessary to calculate expected evolvability is that we need to sample the children of each child, as illustrated in Fig 4.1. The main idea behind

REE-ES is that because there is a large overlap between the distribution of each child, (as can be seen in figure 4.2.), we can reuse the same evaluations multiple times, as part of different child distributions. In order to achieve this we start by evaluating a common set of parameters which are distributed in a way that covers each child distribution well. Then we just select a subset of these evaluations for each child distribution, so the distribution of the selected subset is as close to the child distribution as possible.

This can be achieved by using importance sampling. A famous use of importance sampling is off-policy reinforcement learning [87] where we want to weight experiences coming from a different policy as if they would have come from the current policy. We calculate the sampling probabilities for each point, so when we sample using these probabilities, we get the desired distribution. To calculate these probabilities we can calculate the likelihood ratio which is the Radon–Nikodym derivative of the desired distribution with respect to the original sampling distribution. Equation 4.1 shows how to calculate sampling probabilities, so when we use these probabilities to sample dataset A that has probability density function P_A the resulting sample will have probability density function P_B (Equation 4.2.)

$$P_{sampling} = P_B(A)/P_A(A) \tag{4.1}$$

$$B = sample(A, replace = False, p = P_{sampling}) \tag{4.2}$$

To do an expected evolvability ES update we need to evaluate the evolvability of the population. With REE-ES evaluating the evolvability of the population have the following steps:

1. Sample and evaluate a common sample of points. $A = sample(\mathcal{N}(\theta, 2\sigma), size = c*n)$ We use double the σ so we have good coverage of the children of children distribution. We use the constant c, to determine how many samples we need to achieve good accuracy with the resampling. We discuss how to set c in the next section.

2. Sample a population of children. $B = sample(\mathcal{N}(\theta, \sigma), size = n)$

3. For each child b in population B

- Calculate the resampling probabilities: $P_{sampling} = P_b(A)/P_A(A)$
- Sample a set of pseudo offspring for b: $b_{offspring} = choose(A, size = n, replace = False, p = P_{sampling})$
- Calculate the evolvability of b using the selected subset ($b_{offspring}$) of common evaluations.

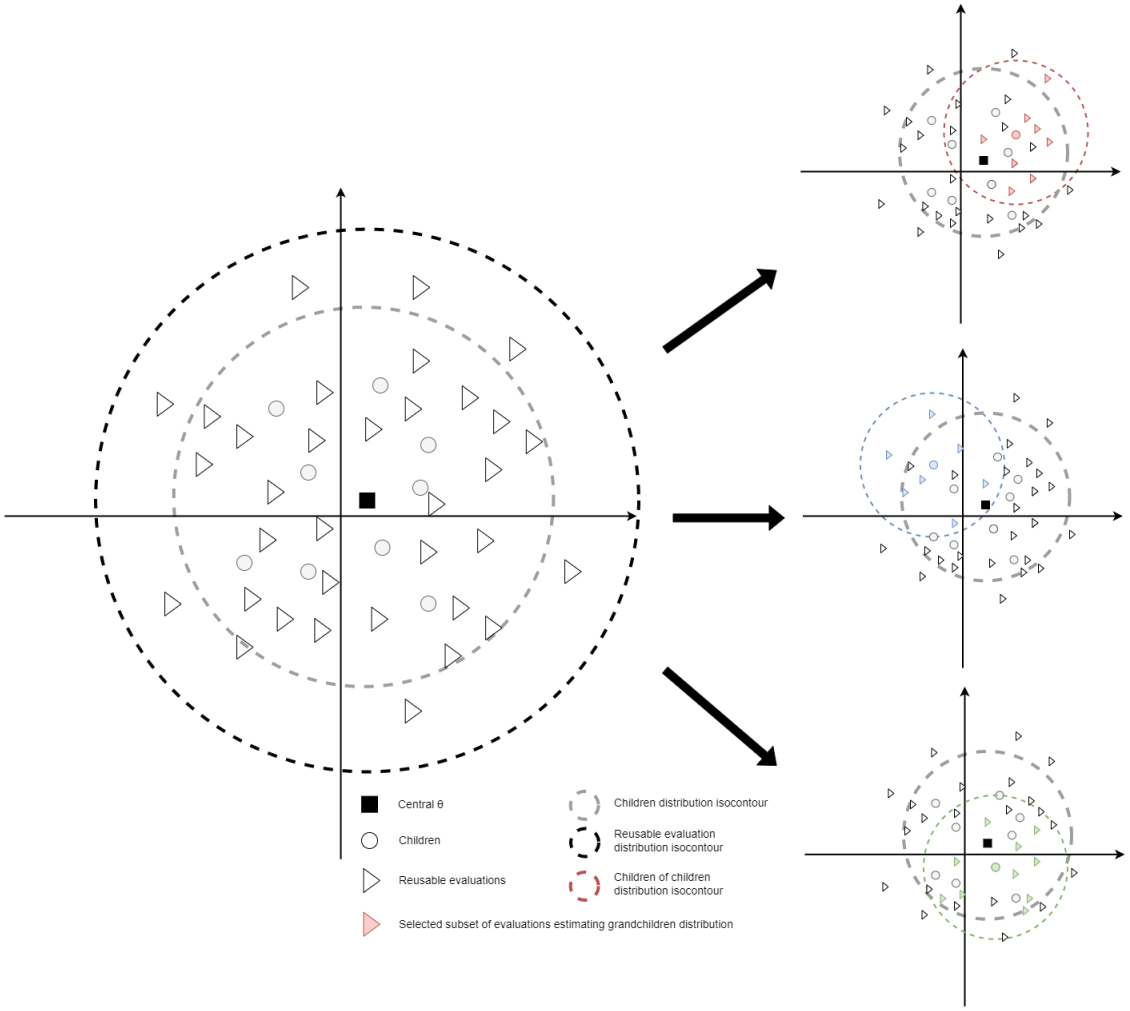


Figure 4.2: This figure illustrates how Resampling EE-ES reuses evaluations multiple times to estimate the evolvability of each child. Because the same evaluations are reused multiple times, Resampling EE-ES can get away with $c*n$ evaluations, (n is the population size, c is a constant not dependent on population size), instead of the n^2 evaluations of the normal EE-ES (4.1).

4.2.4 Number of points required for good quality resampling

There is the question of how many common points we need to sample, so our resampled offspring distribution is close to the original desired offspring distribution. In order to

determine the number of points necessary we can go over a geometry example which demonstrates the problem we are facing here well.

We can imagine two d dimensional spheres, one of them has double the radius of the other. The large sphere represents our common sample, while the smaller sphere represents the distribution we want to create by using a subset of the common points. What is the ratio of their volume? We can use the volume formula for a d dimensional sphere. In equation 4.3. Γ represents Euler's gamma function and R represents the radius.

$$V_d(R) = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)} R^d \tag{4.3}$$

$$\frac{V_d(2R)}{V_d(R)} = \frac{(2R)^d}{R^d} = 2^d \tag{4.4}$$

The reason we calculated the ratio of volumes is that this illustrates the problem of how likely our common points are going to fall into the desired smaller distribution. The bad news is that because the ratio of the distribution volumes changes exponentially with respect to the dimension. This means we need to sample exponentially more points in order to maintain the same coverage of points. This also answers how to set the constant c mentioned in the previous section: $c = 2^d$. While our modified algorithm avoids $O(n^2)$ scaling in population size, it has $O(2^d)$ scaling in dimension size d, which makes it unusable for anything but the smallest dimensional problems ($d < 6$).

4.2.5 Numerical stability of the exponential calculation

When calculating the probabilities, we need to calculate a ratio of log probabilities. This calculation can be very unstable because the exponentiation can cause numeric overflows and underflows. There are several things we can do to increase numerical stability. The first is to rewrite the ratio of exponentiated values into exponentiation of subtraction. The second is to shift the values by their maximum. This way we can avoid numerical overflows with the exponentiation. We can still have numeric underflows, but underflows, in this case, do not cause as large errors, as would overflows, so it is much more important to avoid overflows than underflows.

4.2.6 On the computational cost of the importance sampling

To resample our common evaluations to create the child distributions, we are required to calculate the probability of each point with respect to the distribution of each child. This requires the calculation of the distance between each common evaluation point and each child $O(n^2)$. However, since calculating the distance between parameters is usually much faster than evaluating the problem with the parameters, the resampling calculation is negligible to the evaluations, especially since the distance calculation can be done really fast on a GPU.

4.3 Quality Evolvability

We propose Quality Evolvability (QE), an approach that aims to find individuals with both a diverse and well performing distribution of offspring.

The main motivation for Quality Evolvability is to benefit from an increased ability to evolve in cases where looking for evolvability alone is not sufficient to find solutions to a task. This is similar to the motivation behind Quality Diversity (QD) [2]: to benefit from the divergent, stepping stone finder nature of Novelty Search [88], even in cases when seeking novelty alone is not sufficient to find solutions to a task. In short Quality Evolvability is to Evolvability Search what Quality Diversity is to Novelty Search. Even though Quality Evolvability and Quality Diversity have similar motivations, there are important differences between the two approaches, which are summarized in Table 4.1.

The unique property of Quality Evolvability is that it is forced to become more evolvable because it needs to adapt a single individual to diverse behaviours. While Quality Diversity can benefit from developing evolvability, it is not forced to do so; it can achieve diversity by collecting a set of genetically distant individuals, without any enhanced ability to adapt.

Table 4.1: Comparison of Quality Diversity and Quality Evolvability

Metric	Quality Diversity (QD)	Quality Evolvability (QE)
Approach objective	Find an archive of diverse and well performing individuals	Find an individual with diverse and well performing offspring
Solving deceptive problems	Yes, ever increasing pressure to escape	Yes, constant pressure to escape
Illuminating search spaces	Yes, will explore many ways to solve the problem	No, only explores a small volume of the search space
Encouraging adaptability	No, QD is allowed to use an archive of genetically distant individuals	Yes, QE is forced to adapt a single individual to many different behaviors

Quality Diversity will explore a large section of the search space, aiming to find every possible behaviour that exists. For this reason, it can be used to illuminate search spaces or to find many possible solutions to a problem. Quality Evolvability does not have this property, it focuses its search on a small volume of the search space.

Quality Diversity excels at solving deceptive problems because it will increase the pressure to find novelty until the pressure is high enough to overcome the deceptiveness. Quality Evolvability should also be helpful in escaping deceptive local minima to some degree since it aims to find offspring with diverse behaviours. However, for Quality Evolvability, this pressure is constant and not ever increasing. If the trap is large enough to allow a large amount of diversity, Quality Evolvability is expected to stay trapped.

4.4 Method

In this section, we describe our proposed method, Quality Evolvability ES (QE-ES), which simultaneously selects for both evolvability and fitness. This method is based on ES [32] and Evolvability ES [7]. We discussed these methods in detail in the background chapter 2.

4.4.1 Quality Evolvability ES

Our proposed method Quality Evolvability ES (QE-ES) combines the objectives of ES and E-ES, to simultaneously optimize for both evolvability and fitness. To achieve this we use non-dominated sorting (nd_sort) on the evolvability and fitness objectives. Non-dominated sorting is done the same way as in the well-known NSGA-II [89] algorithm. Individuals are first sorted by which non-dominated front they belong to, and then a crowding metric is used to sort the individual within the fronts. The crowding metric ensures that a diverse set of evolvability-fitness trade-offs are maintained.

For the other versions of ES, rank-based fitness shaping was an optional step, which they found to improve performance [7]. In the case of QE-ES, we naturally work with ranks, because we combine multiple objectives with a nondominating sort. For QE-ES, ranking is not optional anymore.

Algorithm 3: Quality Evolvability ES

Input: Noise standard deviation σ, initial policy parameters θ_0, population size n, gradient optimizer, fitness function F, behaviour characterization BC

for *t = 0, 1, 2, . . .* **do**

Sample $\epsilon_1 ... \epsilon_n \sim \mathcal{N}(0, I)$

$F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, ..., n$

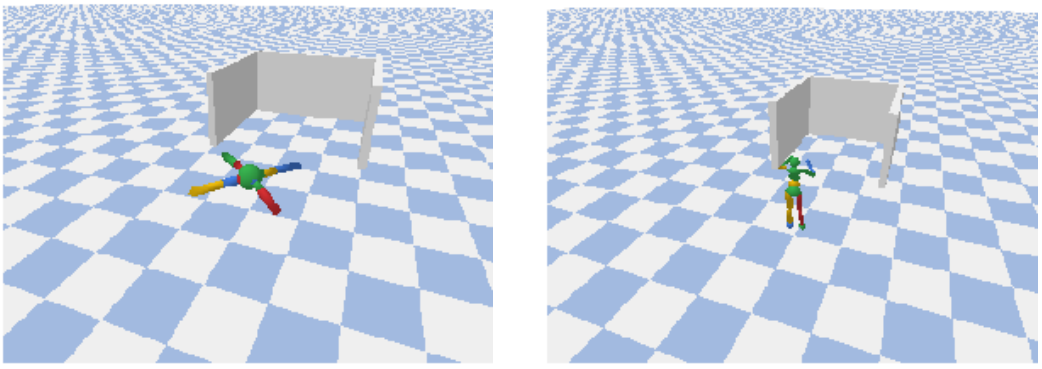
$EVO_i = (BC(\theta_t + \sigma\epsilon_i) - BC_{mean})^2$ for $i = 1, ..., n$

$r = \text{fitness\_shaping}(\text{nd\_sort}(F, EVO))$

$grad = \frac{1}{n\sigma} \sum_{i=1}^{n} r_i \epsilon_i$

$\theta_{t+1} = optimizer(\theta_t, grad)$

end



(a) Deceptive ant task (b) Deceptive humanoid task

Figure 4.3: For the deceptive variant of the tasks, a trap box is put in front of the agent. This obstacle creates a deceptive local optimum. Once the agent discovers how to walk into the box, it cannot improve further without developing the ability to walk around the obstacle. The sides of the trap however make walking around it difficult, requiring the fitness to decrease first. Greedy objective based algorithms are susceptible to such deceptive local optima.

4.5 Experiments

We evaluated our method on the robotics locomotion tasks Ant and Humanoid. An evaluation is comprised of running a full episode until the maximum number of allowed time steps is reached or until the robot falls over. In the default version of the task, fitness is defined as the distance travelled in the x direction, while the behaviour is characterized by the final x and y coordinates of the robot.

We used the more accessible PyBullet implementation of the environments, which has a free software license, rather than the commonly used MuJuCo commercial implementation (MuJuCo received an open-source license after this work was done). An important difference between the implementations is that PyBullet implements the observations dif-

ferently in the case of the Humanoid environment, providing only a $\in \mathbb{R}^{44}$ observation vector compared to MuJuCo which provides $\in \mathbb{R}^{376}$ dimensional observation, containing a more detailed motion state of the robot. This makes the Humanoid experiments more challenging in PyBullet. While with previous experiments using the MuJuCo version of Humanoid, ES was able to find policies that reliably solved the task. In our experiments, ES only finds policies that sometimes solved the task, but not reliably, and not every training run finds such policies. With the Ant environment, there are no significant differences between the implementations, and we could replicate previous results.

Evolution evolves the parameters of a 2-hidden-layer fully connected neural network with 256 neurons for both hidden layers, resulting in a total of 75k parameters for the Ant environment and 81k for the Humanoid environments. The network maps the observations to the actions, with both being real numbers.

For all experiments, we used the same hyperparameter values used in [7]. The population size was 10,000, the noise standard deviation σ was 0.02. We used the Adam optimizer with a learning rate α of 0.01 and L2 regularization of 0.005. We used mirrored sampling and centered rank normalization. The Ant experiments were run for 200 generations, the more difficult Humanoid experiments were run for 800 generations.

ES methods are less sample efficient compared to reinforcement learning methods. In our experiments, a single run consisted of simulating several billion steps. For example, in the case of the successful humanoid run, we have a population size of 10,000, which evolved for 800 generations, where each evaluation takes around 800 time steps. However, ES parallelizes well (linear scaling [32]). We run our experiments on a distributed CPU cluster using 120 cores for each run. With this setting, a Humanoid run took around 1-2 days, depending on the average episode length, while an Ant run took around 6-12 hours. We used the base ES implementation provided by [7].

To test the different properties of our method, we used three modified versions of the environments.

Task 1: Normal Locomotion

For the first experiment, we used the default environments, which we simply refer to as Ant and Humanoid. In this task, evolvability and fitness are aligned. Purely maximizing evolvability will also result in high fitness. This is because evolvability is measured as the variation of the final positions, and the way to maximize evolvability is to learn to walk far in every direction. This includes the forward direction, which means that maximizing evolvability will also maximize fitness.

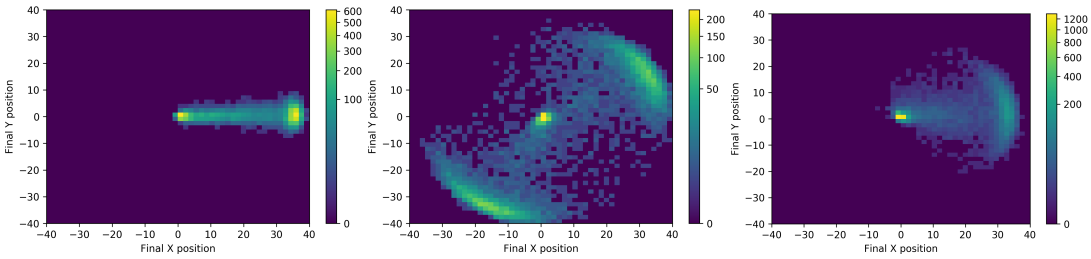
Task 2: Directional Locomotion

For the second experiment, we used modified environments, which we call Directional Ant and Directional Humanoid. Our goal with this experiment is to test our method in a case where evolvability and fitness are unaligned. For these experiments, for each episode, we randomly select a direction from 8 possible directions that are evenly distributed around a circle in 45 degree increments for each episode. The networks receive this direction, represented as the x and y coordinates of a unit vector, which increases the number of observations by 2.

To perform well in this task, the agent needs to turn in the correct direction and walk. Maximizing diversity alone is not enough to achieve high fitness anymore; the agent also needs to learn to walk in the correct direction. Evolvability ES is expected to have zero mean fitness on this task because it equally prefers every direction.

Task 3: Deceptive Locomotion

For the third experiment, we used a deceptive variant of the environments, similarly to previous work [57]. We call these environments Deceptive Ant and Deceptive Humanoid. A U-shaped trap is placed in front of the agent, which allows it to make progress for a while until it runs into a wall (Fig.4.3). The walls on the side prevent the agent from easily going around the obstacle, forming a trap, a local optimum, which is hard to escape. For Deceptive Humanoid, we used the same trap box configuration as in [57], for the Deceptive



(a) Humanoid, ES (b) Humanoid, E-ES (c) Humanoid, QE-ES

Figure 4.4: 2D histogram of the final positions of the population from the last generation for the Humanoid environment, which highlights the different aims of the three algorithms. (a) ES only cares about fitness, and thus only finds policies that walk forward. (b) E-ES only cares about diversity and finds policies that walk in various directions. (c) QE-ES cares about both fitness and diversity, so it finds policies that walk forward in a diverse way.

Ant environment we slightly increased the dimensions of the box from 3 meters to 4 meters in order to make the trap large enough for the physically wider Ant robot.

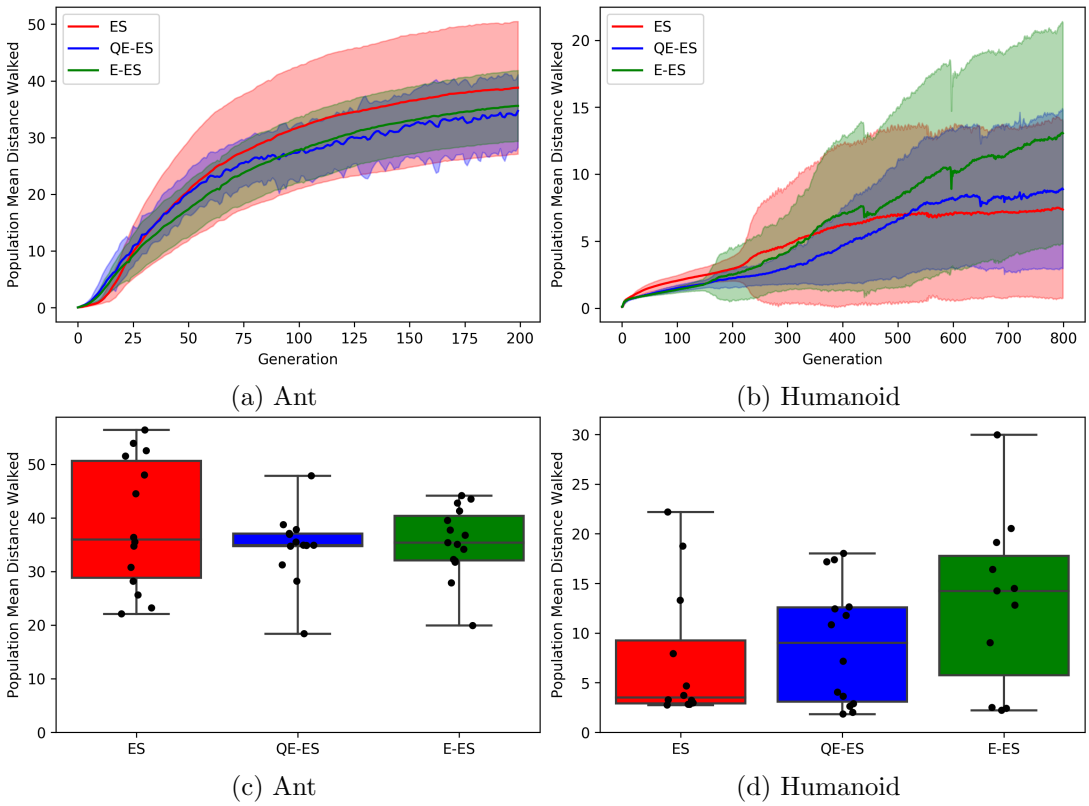
4.6 Results

We use three kinds of plots to present and compare the result achieved with the different algorithms. To show the behaviour of the population in the final generation of a single run, we use a 2d histogram which shows the frequency of the final x, y positions of the robot. The box and strip plots show the mean of the population in the last generation (fitness or distance walked), over repeated runs. Finally, the learning curves show the mean of the population, averaged over repeated runs throughout the generations.

For each task and each algorithm, we repeated the runs at least 10 times. We found that there is a high variance in the results with different random seeds, especially in the case of the more difficult Humanoid task. This agrees with previously reported results in this domain [57].

Task 1: Normal Locomotion

The first experiment was done with the default environments where fitness and evolvability are aligned (Fig 4.5.). We found that adding evolvability does not provide a statistically significant advantage in both environments with regards to maximum fitness. This agrees



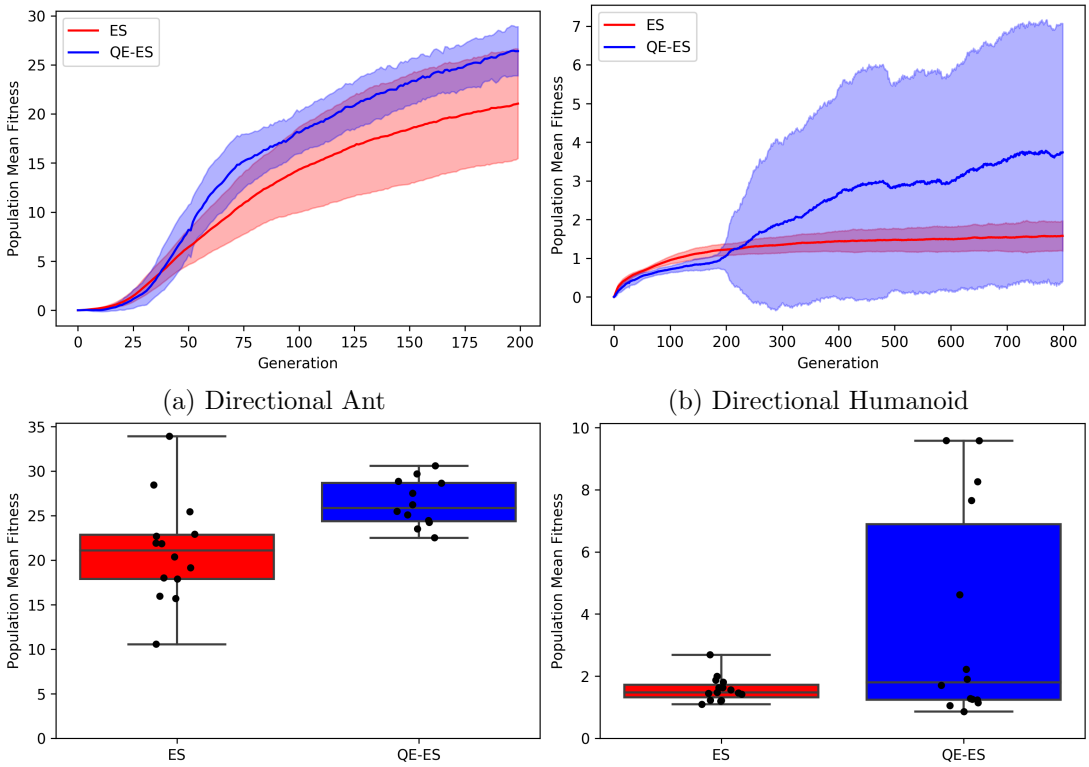
(a) Ant (b) Humanoid

(c) Ant (d) Humanoid

Figure 4.5: Mean distance walked for the Ant (a,c) and Humanoid (b,d) tasks where evolvability and fitness are aligned. The shaded area corresponds to the standard deviation. In this task evolvability and fitness are aligned, looking for one will result in finding the other.

with results presented in the literature previously [7]. Because Quality Evolvability ES is a mix of the two methods, in this aligned task it is expected to perform between the two methods. There is no statistically significant difference in the maximum fitness achieved for each of the methods in this task.

- Ant ES, E-ES: (Median difference: 0.09, Mann-Whitney U Test; p=0.56)
- Ant ES, QE-ES: (Median difference: -0.45 Mann-Whitney U Test; p=0.61)
- Ant QE-ES, E-ES: (Median difference: 0.55, Mann-Whitney U Test; p=0.86)
- Humanoid ES, E-ES: (Median difference: 10.54, Mann-Whitney U Test; p=0.39)
- Humanoid ES, QE-ES: (Median difference: 5.27 , Mann-Whitney U Test; p=0.86)
- Humanoid QE-ES, E-ES: (Median difference: 5.27 , Mann-Whitney U Test; p=0.17)



(a) Directional Ant (b) Directional Humanoid

(c) Directional Ant (d) Directional Humanoid

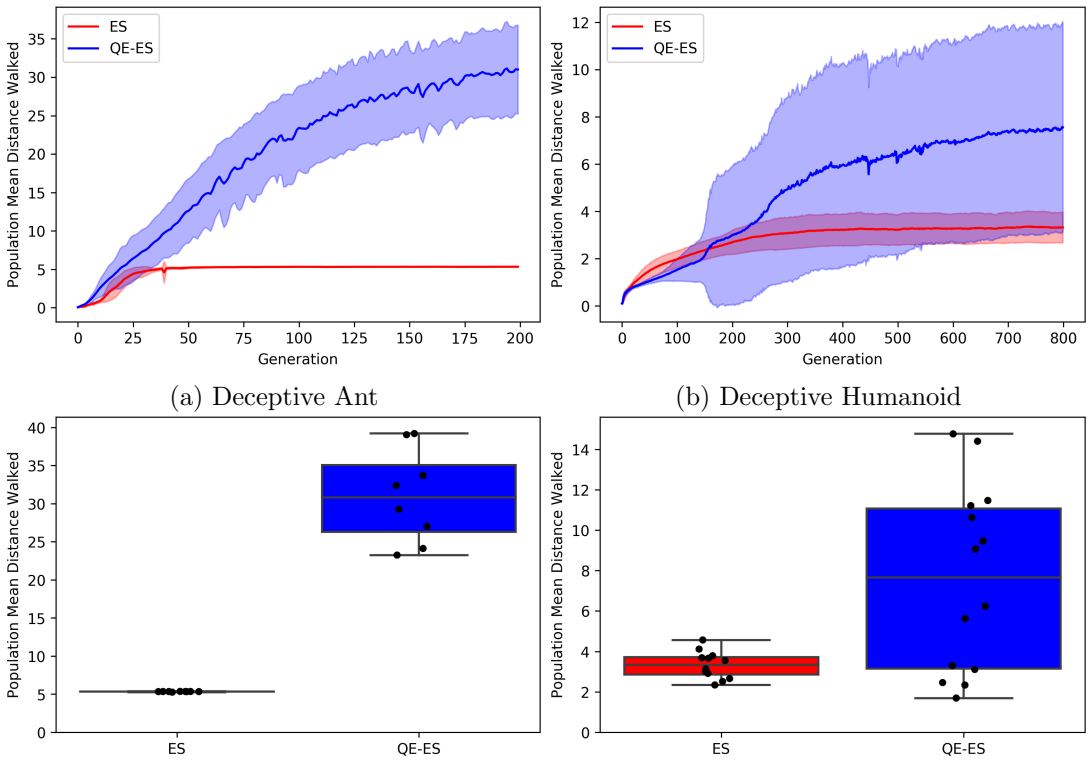
Figure 4.6: Mean fitness for the Directional Ant (a,c) and Directional Humanoid (b,d) tasks, where evolvability and fitness are not aligned. In (a), both algorithms are able to find good policies, but QE-ES receives higher fitness on average. In the more difficult task (b), ES fails to learn to walk and turn in the right direction at the same time, while QE-ES is able to make progress.

When we look at the diversity of the population however, E-ES is expected to have the most diversity, ES the least, while QE-ES is in between the two, which is wat we observe (see Fig. 4.4).

Task 2: Directional Locomotion

The second experiment was done in the directional environments, where evolvability and fitness are no longer aligned (see Fig 4.6). Evolvability ES has zero expected fitness on this problem since it completely ignores the fitness function.

On the Directional Ant task QE-ES achieves higher median fitness than ES (Median difference: 4.70, Mann-Whitney U Test; p=0.0036), on the Directional Humanoid tasks the difference is not statistically significant (Median difference: 0.31, Mann-Whitney U Test; p=0.37). On the Directional Humanoid task, the challenge to simultaneously learn to walk and to turn in the correct direction proved to be too difficult for ES, it always



(a) Deceptive Ant (b) Deceptive Humanoid

(c) Deceptive Ant (d) Deceptive Humanoid

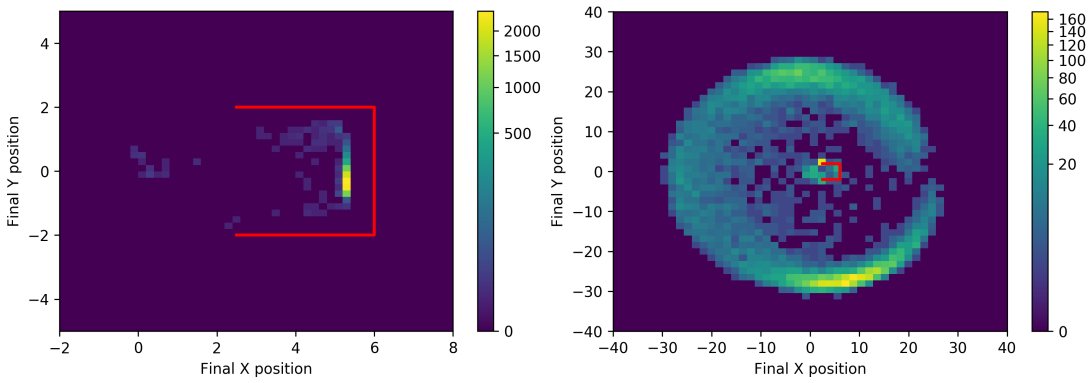
Figure 4.7: Mean distance walked for the Deceptive Ant (a,c) and Deceptive Humanoid (b,d) tasks. For both environments, ES gets trapped every time. For Ant (a,c), QE-ES manages to escape the trap for every run. For the more difficult Humanoid (b,d), QE-ES escapes for the majority of the runs.

gets stuck. In contrast, QE-ES sometimes makes some progress.

These results show that selecting for evolvability can not only result in faster learning, but it can also allow making progress in cases where objective based learning gets stuck.

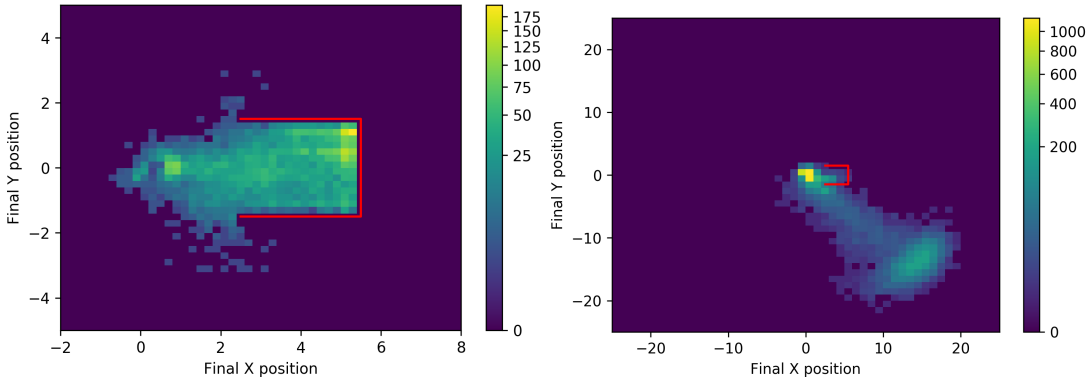
Task 3: Deceptive Locomotion

The final experiment was done on the deceptive variant of the environment, to test which algorithm can deal with deceptive problems (see Fig 4.7 and Fig 4.8). Both with the Deceptive Ant and Deceptive Humanoid environments, ES failed to escape the trap, a result consistent with previous work [57]. QE-ES had no problem escaping the trap in both experiments. For both tasks QE-ES achieves higher median fitness. For Ant: (Median difference: 25.5, Mann-Whitney U Test; p=0.0005), for Humanoid: (Median difference: 5.72, Mann-Whitney U Test; p=0.027) This result demonstrates that Quality Evolvability



(a) Deceptive Ant, ES

(b) Deceptive Ant, QE-ES



(c) Deceptive Humanoid, ES

(d) Deceptive Humanoid, QE-ES

Figure 4.8: 2D histogram of the final positions of the population from the last generation for the Deceptive Ant and Deceptive Humanoid tasks. The red lines represent the walls of the trap, which makes the problem deceptive. For both environments, ES ends up trapped, (a) and (c), while QE-ES is able to overcome the local optimum, (b) and (d). Please note the change in scale.

ES is able to cope with at least some level of deceptiveness, even though it does not have the ever increasing pressure to escape like Quality Diversity algorithms.

4.7 Conclusion

In the introduction chapter we identified three areas in which our knowledge needs to improve in order to realize the vision of ever accumulating evolvability. These are providing diverse evolvability data, creating models which have a high capacity to learn evolvability and discovering algorithms which can drive learning evolvability.

In this chapter, we provided two novel evolvability algorithms. We presented Resamping Expected Evolvability ES, while an interesting approach, it has limited practical application due to the poor scaling in the problem dimension. We have also presented

Quality Evolvability ES, a technique to simultaneously select for evolvability and fitness. QE-ES allows us to benefit from evolvability in cases when evolvability and fitness are not aligned. While our experiments demonstrated that evolvability can increase the performance of evolution in a single environment, it is yet to be determined whether the learned evolvability is general enough to be useful in different environments and with different behaviour characterizations. To truly test the generalization capability of evolvability, it is necessary to use more complex environments with multiple different tasks, such as Meta-World [90].

Quality Evolvability and Quality Diversity are not mutually exclusive approaches. There is no reason which prevents the combination of the two approaches to combine the benefits, by looking for an archive of diverse and evolvable individuals. In the next chapter, we explore this direction.

Chapter 5

Evolvability Map Elites

5.1 Introduction

In this thesis, we set out the goal to create evolution of evolvability algorithms. In this chapter, we introduce the Evolvability Map Elites algorithm. The original Map Elites [3] algorithm aims to explore the search space by finding a diverse set of highly fit individuals. Evolvability Map Elites aims to find a diverse set of not just fit, but both fit and evolvable individuals. We have two main motivations for choosing to create evolvability map elites.

First, we would like to tap into the synergistic relationships between quality, diversity and evolvability. Previous algorithms tap into the synergies between quality and diversity [2], and between quality and evolvability [5]. There is no algorithm yet which attempts to utilize the synergies between all three.

Second, recent advances in Evolutionary Strategies algorithms [7, 32, 91] allow us to calculate all kinds of metrics and gradients without needing extra evaluations. With only a single sample of evaluations, we can calculate adaptiveness, innovativeness, evolvability and the gradient to all of these. Since we already need these evaluations for calculating fitness, it would be a waste not to utilise these evaluations for other purposes as well. The largest benefit comes from being able to calculate evolvability gradients with a single sample. Before Evolvability ES [7], calculating gradients to evolvability required evaluating $O(n^2)$ (n is the population size) samples [26], and now we can get it for free if we already need the evaluations for fitness.

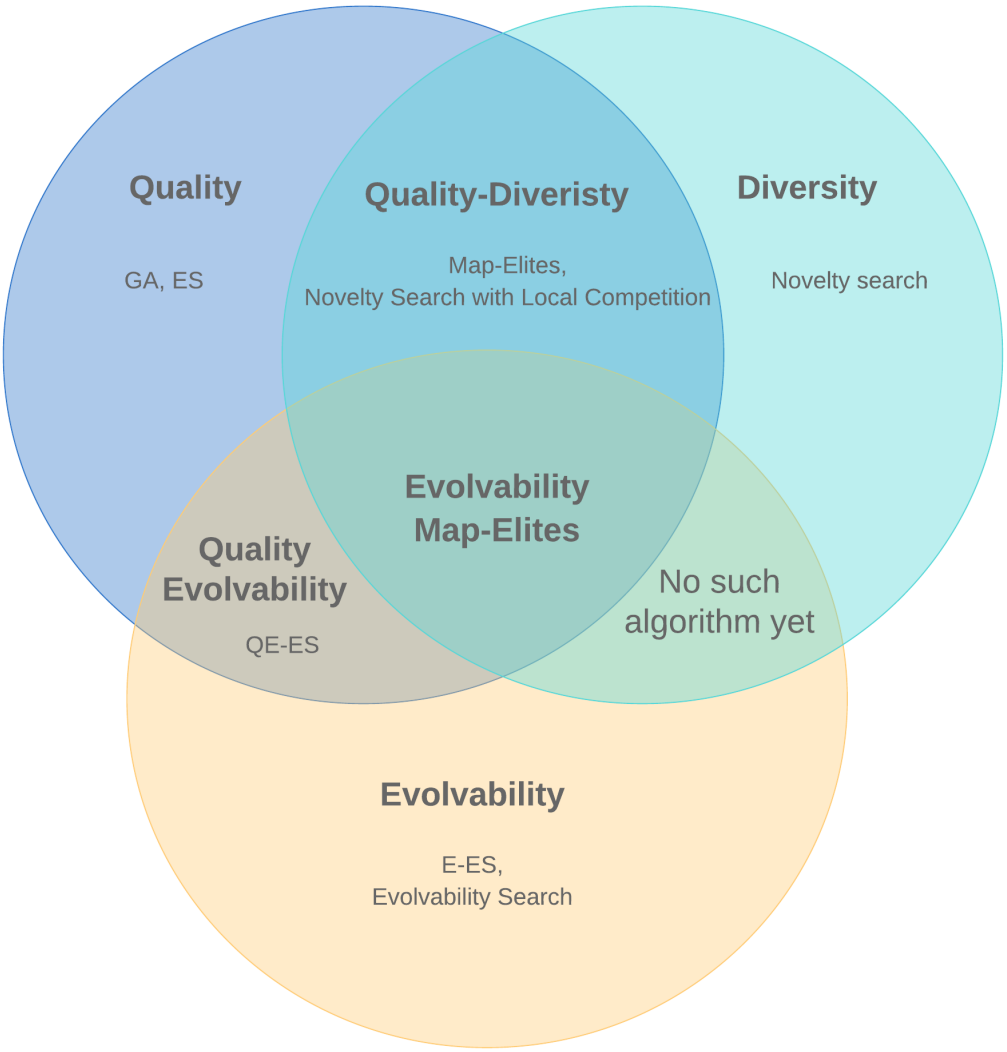


Figure 5.1: Family of algorithms, focusing on quality, diversity, and evolvability. The intersection of quality and diversity is well studied [2] with famous Quality-Diversity algorithms such as Map Elites [3] and Novelty Search with Local Competition [4]. In chapter 4. we explored the intersection of quality and evolvability with the new algorithm Quality Evolvability ES [5]. The topic of this chapter is the intersection of all three, with our proposed algorithm: Evolvability Map Elites.

5.1.1 Quality, Diversity, Evolvability

Before we discuss the benefits of combining quality, diversity and evolvability, first we go over the definitions. Quality is a measure of the performance (fitness) of a solution on a task. Diversity is a measure of how many behaviorally different solutions we found, and how different they are. Evolvability is the ability to generate good quality (adaptive, innovative, diverse) phenotypic variation.

There are good reasons for desiring either quality, diversity or evolvability on their own. It is obvious why we want quality, we want solutions which are good at performing their task. Diversity can be useful to find a repertoire of solutions [92], so if one solution stops working we can try others which use a different strategy or behaviour. Or diversity can help us create a wide variety of levels [93] for computer games which do not get boring fast or even a wide variety of game designs [94]. Evolvability can enable fast adaptation to new tasks [7]. Optimizing for evolvability can help us utilize indirect encoding, by facilitating the discovery of reusable modules and building blocks [8] (there are more details on the relationship between evolvability and indirect encoding in chapter 2).

We argue that even if we only care about one of these objectives, it is worth looking at the others. It might seem counterproductive, to simultaneously look for multiple objectives, leading to sub-optimal results in all of them. However, in the process of looking for either one of these aims, we might find stepping stones which can be utilized when we optimize for the other aims.

There are many examples of such synergistic relationships between these objectives reported in the literature. One of the most famous examples is the success of novelty search [36]. In certain deceptive tasks, where it is easy to get stuck in a local optimum, looking for novelty alone can lead to the discovery of better quality solutions compared to directly optimizing for quality. This result inspired the field of quality diversity (QD) [2]. QD algorithms simultaneously look for quality and diversity. The most famous QD algorithms are map elites [92] and novelty search with local competition [4]. In the previous chapter, we presented the Quality Evolvability (QE) algorithm QE-ES. In the case of QE, we simultaneously look for quality and evolvability. In the previous chapter, we showed that for certain versions of the task, adding the evolvability objective leads to a higher

quality compared to only optimizing for quality.

Fig 5.1 shows the Venn diagram of algorithms with the aims of quality, diversity and evolvability. In the previous chapter, we examined the previously unexplored intersection of quality and evolvability. In this chapter, we examine the intersection between all three aims, by creating Evolvability Map Elites. We hope that by simultaneously looking for all three, we can find even more stepping stones and reach better performance in all of the objectives, compared to only looking for one or two aims at a time.

Although we see many examples of synergies in the literature, there are also examples when there is a tradeoff between quality, diversity and evolvability. For example in the case of novelty search, a small change in the maze navigation environment (one of the walls removed from the maze) results in the robot aimlessly wandering outside the maze and never learning any navigation skills (avoiding bumping into walls). This is because it is easier to achieve novelty outside the maze, where no such skills are necessary [36]. Another example can be found in the previous chapter, using evolvability beside quality with ES increased the quality of the solutions for the deceptive task, but not for the basic task. For the basic task, there was a tradeoff between quality and evolvability. Whether there is a tradeoff or a synergy between the objectives is task and behaviour characterization dependent. It is also worth mentioning that because of bias towards publishing positive results [95] the literature likely presents an overoptimistic picture of how often the relationships between these aims are synergistic.

5.2 Background

In this chapter, we create variants of the map elites algorithm which utilizes direct search for evolvability. In this section, we discuss the inner workings of the original map elites algorithm.

5.2.1 MAP elites

Map elites [92] is a quality diversity algorithm. The aim of map elites is not just to find a single highly fit solution, but a diverse set of them. This diversity is quantified with the

help of a behavioural characterization function. We discussed behaviour characterization functions in the previous chapter (4) in detail, as evolvability ES and quality evolvability ES also uses behavioural characterization functions.

Map elites go about creating diversity by partitioning the behaviour space into cells. Each cell acts as a protected niche. Once a cell is populated we keep around that individual called elite, until a better-performing individual comes around who belongs to the same cell (exhibits the same behaviour).

A single iteration of map elites consists of the following steps:

- select a parent from the existing elites in the map.
- apply a parameter update to the parent to create a child
- attempt to insert the child into the map

For the insertion, the behaviour of the child determines which cell in the map it belongs to. The insertion decision is made using the following rules:

- If the cell of the individual is empty, insert the individual in the cell
- If the cell is not empty, but the fitness of the current individual is higher than the fitness of the elite in the cell, replace the elite with the current individual.

5.2.2 Behaviour space partitioning

One component of all map elites algorithms is behaviour space partitioning. The most common way to partition the space is by creating a grid [92]. A grid works well for small dimensional spaces, however, if the behavioural characterization is high dimensional, the grid size would explode. To be able to use map elites with high dimensional behaviour spaces, the Centroidal Voronoi Tessellations (CVT) partitioning method was proposed [96]. CVT divides the space into a fixed number of cells, irrespective of the number of dimensions.

5.2.3 Parameter update

Another component is the parameter update. The original map elites used simple mutation. Since then the field progressed a lot and the line between evolutionary computation and gradient-based optimization blurred. Several parameter updates have been proposed that utilized some kind of gradient-based updates. For reinforcement learning problems, we can use policy gradient as the parameter update [97]. Or in the case of Policy Manifold Search [98], a policy manifold is learned from the elites with the help of an autoencoder. Then the parameter update can be done in the latent space, which can be decoded back into parameter space.

There were also variants of MAP Elites created which use more efficient black box optimization algorithms as parameter update. One such algorithm is CMA-ES, resulting in CMA-ME [99]

Another development in the parameter update component of Map Elites is Multi Emitter Map Elites [100], where a bandit algorithm dynamically selects between various parameter update methods in order to bias selection towards the most succesful type of emitter.

One of the most interesting developments in the area of map elites algorithms from our perspective is using an ES update as the parameter update component of map elites. The resulting algorithm is called Map Elites ES [91]. ES is an optimization method which allows us to estimate a gradient for the expected fitness of offspring by sampling points from a normal distribution around the parent individual. We discussed ES in detail in the chapter 3, including evolvability ES, the variant with the objective to maximize evolvability.

The main reason Map Elites ES is interesting is that it allows us to calculate all kinds of extra metrics and updates almost without the need for additional evaluations (see next section 5.3.1 for more details).

Map Elites ES [91], uses two kinds of ES updates. When using fitness based ES update, it is called exploit mode, and when using novelty based ES update, it was called explore mode. The authors argue that there are different kinds of pressures in explore

and exploit mode. Even in exploit mode, where the objective of the parameter update is fitness, the archiving decision provides some pressure for exploration, as novel individuals are kept even if they have bad performance (if their behaviour cell is empty). And in turn in explore mode, where the objective is innovativeness, the individuals are only kept if they are the elite of their cell, fitness also matters.

5.2.4 Archiving decision

Another component of map elites is what kind of map is used to store the elites and what metric is used to make the insertion decision.

Most previous versions of Map Elites used a single metric for the insertion decision which was evaluation fitness, and used a single map, where each cell in the map contained the individual who has the highest fitness for that behaviour

Some notable exceptions are Multi Objective MAP-Elites (MOME) [101] and Multi-Container AURORA [102]. MOME is applied to multiobjective optimization problems, and in each cell a pareto front of individuals with various tradeoffs between the objectives is maintained. In case of Multi-Container AURORA there are multiple maps, each one using a different behaviour descriptors.

Our aim in this chapter is to create an evolvability version of map elites. For this, we created a multiobjective version of map elites. In our version the multiple objectives MAP-Elites the objectives are the different aspects of evolvability. This is different from MOME where the multiple objectives come from the multiobjective task. We will discuss what kind of changes were necessary to Map Elites to be able to utilize multiple metrics in the next section.

5.3 Method

In this section, we describe the algorithm components we added to allow map elites to also optimize for evolvability besides quality and diversity. We provided python like pseudocode for each of these components.

```python
def evo_map_elites_step(evo_map_archive,novelty_archive):
    # select type of update we want to do
    parameter_update_mode = np.random.choice(config["parameter_update_modes"])

    # select a parent
    parent_theta,parent_evaluations = parent_selection(evo_map_archive,
    parameter_update_mode)

    # do gradient step. NOTE if this individual was a parent before we should recalcualte
    gradients with new random seed
    new_theta = parameter_update(parent_theta,parent_evaluations[f"grad_{
    parameter_update_mode}"])

    # calculate metrics and gradients
    new_evaluations = calculate_all_metrics_and_gradients(new_theta)

    # Whether or not we insert it in the map, we add it to the behavior archive
    novelty_archive.add_to_archive(new_evaluations["behavior_characterization"])

    # try insert into the map
    try_insert_in_archive(evo_map_archive,new_theta,new_evaluations)
```

Listing 5.1: Evolvability Map Elites algorithm

5.3.1 Computational considerations

When we calculate the evaluations for an ES update, we can calculate all kinds of extra metrics and updates without the need for extra evaluations. These additional metrics include adaptiveness, innovativeness and evolvability, and the gradients for these three objectives (see Tab 5.1.). Pseudocode of calculating all these metrics and gradients are shown in Listing 5.3.

```python
def kde(bcs):
    """ Helper function for kernel density estimation"""
    k_sigma = config["ENTROPY_CALCULATION_KERNEL_BANDWIDTH"]
    pairwise_sq_dists = scipy.spatial.distance.squareform(scipy.spatial.distance.pdist(bcs
    , "sqeuclidean"))
    k = np.exp(-pairwise_sq_dists / k_sigma ** 2)
    return k

def calculate_novelty(bcs,novelty_archive):
    """ Returns the mean distance of each bc to the k neerest neighbor in the novelty
    archive"""
    nn_model = sklearn.neighbors.NearestNeighbors(n_neighbors=config["NN_K"], algorithm='
    ball_tree', metric='euclidean')
    nn_model.fit(np.stack(novelty_archive))
    distances, indicies = nn_model.kneighbors(bcs,n_neighbors=config["NN_K"])
    return np.mean(distances,axis=1)

def rank_based_score(scores):
    """Assigns a weigth for each score between -0.5 and 0.5 based on their rank"""
    sorted_indicies = np.argsort(scores)
    rank_based_score = np.zeros(len(sorted_indicies))
    rank_based_score[sorted_indicies] = np.linspace(-0.5,0.5,len(sorted_indicies))
    return rank_based_score

def nd_sort_individuals(individuals,metrics):
    scores = np.zeros(len(metrics),len(individuals))
```

```
    for metric_i,metric in enumerate(metrics):
        for i,(params,eval) in enumerate(individuals):
            if metric == "innovativeness":
                # for a fair comparision, innovation must be recalculated with current novelty archive
                eval["innovativeness"] = recalculate_innovativeness(eval["child_evaluations"])
            scores[metric_i,i] = eval[metric]

    sorted_indicies = non_dominated_sort(scores)
    return sorted_indicies
```

Listing 5.2: Helper functions for Evolvability Map Elites

```

def calculate_all_metrics_and_gradients(theta,evaluate_fun,novelty_arhive):
    """
    Args:
        theta: np.array[num_parameters]
        evaluate_fun: function which evaluates parameters on a task. Returns fitness:float and behavior characterizarion:np.array[bc_dimension]
        novelty_arhive: np.array[archive_size,bc_dimension]
    Returns:
        dictionary with metrics and gradients for theta
    """
    N = config["NUM_SAMPLES"]
    sigma = config["ES_sigma"]

    # evaluations
    evaluation_fitness,evaluation_bc = evaluate_fun(theta)
    mutations = np.random.randn(len(theta),N) * sigma
    child_fitnesses, child_behavior_characterizations = [],[]
    for child_i in range(N):
        child = theta + mutations[child_i]
        fitness,bc = evaluate_fun(child)
        child_fitnesses.append(fitness)
        child_behavior_characterizations.append(bc)

    # fitness
    adaptability = np.mean(child_fitnesses)
    es_grad_weights = rank_based_score(child_fitnesses)
    grad_es = -np.matmul(es_grad_weights,mutations) / N / sigma

    # behavior distribution entropy
    k = kde(child_behavior_characterizations)
    p = k.mean(axis=1)
    entropy = -np.log(p).mean()
    entropy_grad_weights = - np.log(p) - (k / k.mean(axis=0)).mean(axis=1)
    grad_evolvability_es = -np.matmul(entropy_grad_weights,mutations) / N / sigma

    # behavioral novelty
    novelties = calculate_novelty(child_behavior_characterizations,novelty_arhive)
    innovativeness = np.mean(novelties)
    novelty_grad_weights = rank_based_score(novelties)
    grad_novelty_es = -np.matmul(novelty_grad_weights,mutations) / N / sigma

    result = {}
    result["evaluation_fitness"] = evaluation_fitness
    result["behavior_characterization"] = evaluation_bc

    result["adaptability"] = adaptability
    result["evolvability"] = entropy
    result["innovativeness"] = innovativeness

    result["grad_es"] = grad_es
    result["grad_evolvability_es"] = grad_evolvability_es
    result["grad_novelty_es"] = grad_novelty_es
```

```
53
54      result["child_evaluations"] = (child_fitnesses,child_behavior_characterizations,
        mutations)
55      return result
```

Listing 5.3: Function for calculating all metrics and gradients used in Evolvability Map Elites

In reality, the situation is a little more complicated. The same evaluations can indeed be used to calculate the metrics and the gradients. The issue comes from when we need this information. When a new individual is created, first we need to evaluate the metrics, to decide if an individual is going to be inserted into the map. Later, only if the individual is inserted and if it is selected as a parent, do we need to calculate the gradient for an update. So to access the information at the right time, we need to do a look-ahead evaluation. Because of the uncertainty of whether the individual is going to be inserted in the map and selected as a parent, these look-ahead evaluations might be wasted (never used for calculating a gradient). In the worst case, this can double the necessary computation. The cost for the lookahead computations however is much lower in reality for two reasons.

First, we can cheaply cache the evaluation results. We need to store the fitnesses (scalar), the behaviour characterizations (low dimensional number) and the mutation vectors. All this information is needed to calculate the gradients. The mutation vectors are huge, they require the memory of model size times population size. However, it is enough if we store random seeds (as was used in [77]) or indices to a random table (as used in [32]) instead, and recalculate the mutation vectors when needed. The random table indices were first introduced as a way to save network bandwidth when running ES on a CPU cluster, and the random seed method was introduced for the same reason for the GA algorithm. However, in this work we use random table indices for a different purpose, to reduce the memory footprint of caching. This is because we want to store many of these evaluations for a long time, as they might be needed for gradient calculation if the parent is selected in the future.

The second reason is that we do not need to select a completely new parent every generation. As in the original Map Elites ES paper [92], we can continue to iterate on the same lineage for a fixed number of steps (e.g. 10). This means we can be sure that we can use the look-ahead evaluations right away to calculate the next gradient update. Using

this method, in the worst case we only waste 10% of the evaluations. One important detail for implementation is that for an individual which is selected as a parent repeatedly, we must not use the cached evaluations more than one time (for the same update mode) since that would lead to an identical child.

In conclusion the real extra cost of doing these lookahead evaluations is less than 10%.

5.3.2 Optimizing for evolvability

One way we can introduce evolvability into map elites is to directly optimize for evolvability with the parameter update component of map elites. Pseudocode for the parameter update is provided in Listing 5.4. There are various aspects of evolvability, as we discussed in detail in chapter 2.

```
def parameter_update(theta,grad):
    theta = torch.nn.Parameter(torch.from_numpy(theta))
    optimizer = torch.optim.SGD([theta],lr=config["ES_lr"],weight_decay=config["
    ES_L2_COEFF"])
    theta.grad = -grad # torch optimizer is minimizing, we want to maximize, multiply by
    -1.
    optimizer.step()
    return theta.detach().numpy()
```

Listing 5.4: Parameter update using gradient descent for Evolvability Map Elites. The gradients are provided by the calculate_all_metrics_and_gradients function.

One aspect of evolvability is adaptiveness (expected fitness of offspring). The normal ES update optimizes for adaptiveness. This is the update Map Elites ES use in exploit mode. Another aspect of evolvability is innovativeness (the expected novelty of offspring). Novelty ES optimizes for innovativeness. This is the update Map Elites ES use in explore mode.

Finally, the third aspect of evolvability is the behavioural diversity of offspring. We call this aspect evolvability, which is very confusing, but it has historical reasons. This is the evolvability definition which the first two algorithms directly optimizing for evolvability used, evolvability search [26] and evolvability ES [7]. There are multiple ways to quantify the behavioural diversity of offspring [7]. One way is to calculate the variance of the behaviour of the offspring distribution. This is simple to calculate, but it favours extreme behaviours, as the way to maximise variance is to occupy the extremes. Another way to quantify evolvability is as the entropy of offspring behaviour distribution. This method

Metric name	Metric fromula	ES update
Evaluation Fitness	$F(\phi)$	No ES update exists
Novelty	$Nov(\phi)$	No ES update exists
Evolvability (entropy)	$Evo(\phi) = -\mathbb{E}_{\theta \in \mathcal{N}(\phi,\sigma))}(-log(p(B(\theta))))$	Evolvability ES
Expected fitness (adaptiveness)	$\mathbb{E}_{\theta \in \mathcal{N}(\phi,\sigma)}(F(\theta))$	ES
Expected novelty (innovativeness)	$\mathbb{E}_{\theta \in \mathcal{N}(\phi,\sigma)}(Nov(\theta))$	Novelty ES
Expected evovlability	$\mathbb{E}_{\theta \in \mathcal{N}(\phi,\sigma)}(Evo(\theta))$	No (too expensive)

Table 5.1: Metrics, formulas and ES updates. ϕ is the current parameters, while θ are the offsprings. σ is the ES σ. B is the behaviour characterization function. For evolvability we used the formula for the max entropy variant [7] which we used in this chapter. For Evolvability Map Elites, we used the metrics marked with blue as selection criteria and we use the ES updates marked with green as parameter updates. The interesting thing about ES is that all 3 metrics (evolvability, adaptiveness, innovativeness) and the 3 corresponding updates can be calculated using the same evaluations.

encourages a uniform distribution for all behaviours. We chose to use the entropy version, as it avoids only rewarding extreme behaviours. The entropy version has some extra steps, as we need to do a kernel density estimation, to estimate the probability of each offspring. With evolvability ES, we introduced a third mode to Map Elites ES besides exploit and explore. We call this mode evolvability building.

Our original aim in this chapter is to create an algorithm which is able to utilize the synergies between objectives. In order to do this, we need a way to combine multiple modes of updates. In the original Map Elites ES algorithm this was done by switching between modes every couple of iterations (between explore and exploit). In the previous chapter, we proposed a method to combine different ES objectives into a single ES update [5]. To do this, we combined adaptiveness and evolvability into a single update by doing a nondominated sort to rank the individuals in the ES update. We did some initial experiments using this kind of combined updates but found that they do not seem to make a large difference compared to switching update modes. We intend to examine the question in more detail in the future, however, for the experiments discussed in this chapter, we used switching modes.

5.3.3 Archiving based on other metrics besides fitness

In most previous versions of map elites, the decision whether an individual is inserted into the map as the elite of its niche was based on one metric: its evaluation fitness. Because we do lookahead evaluations (as discussed in the previous section), we now have the ability to cheaply calculate and select for other metrics. These other metrics include adaptiveness, (expected fitness of offspring), innovativeness (expected novelty of offspring) and evolvability (entropy of offspring distribution) (see Tab 5.1).

Of course, there are good reasons for selecting for evaluation fitness. One of the goals of the algorithm is to find high fitness individuals. However, we argue that there are also reasons to select for other metrics, and the disadvantage of not selecting for evaluation fitness can be mitigated. This mitigation strategy will be discussed in the next section.

One reason for selecting for these metrics is that aiming for these metrics is a worthy goal in itself. By having more adaptive, innovative and evolvable individuals as parents, we hope to generate more successful children. By using them as the metric for archiving decision, we can add pressures to the algorithm towards increasing these metrics.

Another reason is that in the case of Map Elites ES, the optimization objective is not aligned with the selection objective. The objective of ES is not fitness of the individual, but the expected fitness of the offspring of the individual. This created a situation where the decision which individual to keep in the elite archive was based on a different metric compared to the objective of the parameter updates. We hypothesise that selecting for adaptiveness instead of evaluation fitness, and so avoiding this kind of contradicting objectives for the archiving decision and the optimization will result in faster learning. The same is true with novelty ES and evolvability ES, where they optimize for innovativeness and evolvability. We also align selection with optimization for these ES updates as well.

5.3.4 Elite evolvers and elite performers

In the previous section, we introduced selecting for metrics other than evaluation fitness, which might make sense for the algorithm, however, we lost the ability to keep track of the

true elite performers. This is one of the goals of map elites, to find the elite performers, so by doing this change, we made the algorithm worse in one of its goals.

To remedy this problem, we introduced a two map system. To make it possible to do archiving based on evolvability metrics while still keeping track of the highly fit individuals, we introduce a separate elite performer map and elite evolver map. The elite performer map keeps track of the highly fit individuals, just like in previous map elite variants. The new elite evolver map keeps track of the individuals which are best in the evolvability metrics (adaptiveness, innovativeness, evolvability). On each iteration of the algorithm, we always try to insert the current individual into both maps, however when it comes to parent selection, we only select from the elite evolver map.

Our elite performer map is similar to the previously used passive archive [101] in the sense that the passive archiver does not influence the algorithm, just logs solutions. Passive archive was introduced to do fair comparison between MOME and MAP-Elites on grid size dependent metrics like grid coverage. In our case the passive elite performer archive is necessary because we are not selecting for the best performers, so without an elite performer archive we might forgot what solution was the best performer during the run.

Introducing a second map does incur a memory cost since we need to store two maps in the memory. In practice, we use a smaller evolver map, so this memory cost is not significant. The reason for using a smaller evolver map is to ensure the lineages are long enough for ES to make progress. We discuss the lineage length issue of Map Elites ES in the next section.

Another reason the memory cost is not significant is that there is some overlap between the maps, and we can just store a reference to the same elite in both maps.

5.3.5 Archiving decision based on evolvability

Our original aim for introducing evolvability to map elites is to be able to utilize any synergies between quality, diversity and evolvability. In the previous section, we discussed how we can optimize and select for evolvability. However to fulfil our goal we want to keep the quality and diversity components of map elites as well. To achieve this we cannot just

replace fitness with evolvability, we need to introduce evolvability while still caring about fitness and diversity.

We came up with two methods to create this kind of multiobjective map elites. One is called multi map, the other is called nondominated map (nd map). The pseudocode for the algorithm for the archiving decision is provided in Listing 5.5.

```
def try_insert_into_cell(cell,metric,theta,eval_result):
    if cell.is_empty():
        cell.insert(theta,eval_result)
    else:
        curr_elite_eval_result = cell.get_elite_eval()
        if metric == "innovativeness":
            # for a fair comparision, innovation must be recalculated with current novelty archive
            curr_elite_eval_result["innovativeness"] = recalculate_innovativeness(curr_elite_eval_result["child_evaluations"])

        if eval_result[metric] > curr_elite_eval_result[metric]:
            cell.insert(theta,eval_result)

def try_insert_into_non_dominated_cell(cell,metrics,theta,eval_result):
    if cell.is_empty():
        cell.insert(theta,eval_result)
    else:
        elite_front = cell.get_elite_front()
        elite_front.append(theta,eval_result)
        if len(elite_front) > config["ND_FRONT_SIZE"]:
            # do nondominated sort and drop the last.
            sorted_indicies = nd_sort_individuals(elite_front,metrics)
            cell.set_elite_front(elite_front[sorted_indicies][:config["ND_FRONT_SIZE"]])

def try_insert_in_archive(evo_map_archive,theta,eval_result,config):
    cell_i = evo_map_archive.get_cell_index(eval_result["behavior_characterization"])

    if config["archive_mode"] == "single_map":
        try_insert_into_cell(evo_map_archive[cell_i],config["single_map_archiving_metric"],theta,eval_result)

    if config["archive_mode"] == "multi_map":
        for metric in enumerate(config["multi_map_archiving_metrics"]):
            try_insert_into_cell(evo_map_archive[cell_i,metric],metric,theta,eval_result)

    if config["archive_mode"] == "non_dominated_map":
        try_insert_into_non_dominated_cell(evo_map_archive[cell_i],config["nd_map_archiving_metrics"],theta,eval_result)
```

Listing 5.5: Insertion decision algorithm for Evolvability Map Elites. For single map, the map ia a list of cells each of which contain an elite. For multi map the map is a list of single maps, one map for each archiving metric. For non dominated map, the map is a list of cells each of which contain a nondominated front of elites.

In the case of multi map, instead of using a single map, we use one map for each metric. When we attempt to insert a new individual into the map, we try to insert it in all of the maps, testing whether the individual is elite with respect to the corresponding metric. With multi map, we only care about the eliteness in a single metric, we completely ignore the performance in other metrics. Of course, this makes our map larger, for example

when we use three metrics, our map is three times larger.

Multi map is similar to Multi-Container AURORA [102]. The main difference is in Multi-Container AURORA there is a separate map for every behaviour characterization, while in our multi map archive we have a separate map for each archiving metric.

In the case of nondominated map, we only have one map, but in each cell, we maintain a nondominated front of elites. The aim is to maintain various tradeoffs between the metrics. When we try to insert an individual, we retrieve the elites found already in the cell, then do a nondominated sort with all of them, and keep the first n. In our case we used $n = 5$, so we could maintain a reasonable set of tradeoffs while not making the size of the map too large.

Nondominated map is using the same method as MOME [101], the difference being that MOME is applied to multiobjective optimization problems, while our method applied to normal optimization with a single objective, and we create the multiple objectives from various aspects of evolvability.

5.3.6 Parent selection

The original map elites used uniform selection for deciding which individual from the map becomes the parent of the next generation. Since then researchers experimented with all kinds of biased parent selection, where some individuals have a higher chance of being selected for reproduction than others. One such method is curiosity based selection [103] where the selection weight depends on the success of the individual to generate offspring which is successfully inserted in the map. As long as an individual keeps generating interesting offspring, we keep selecting it. An interesting result showed that selecting young individuals works similarly well as curiosity based selection. [104], because it mimics the population of novelty search. Another option is to select parents proportionally to their fitness, the higher fitness individuals have, the higher chance they have to be selected [91].

When only a single metric is used in the map, the selection strategy is straightforward. We just use the metric for weighting selection which we use for the map insertion decision. In this work, we introduce versions of map elites where the goal is to find a

diverse set of individuals with different tradeoffs between quality, evolvability and innovativeness. In this version, we no longer have a single metric in our map. In this kind of multiobjective algorithm, the parent selection should also be based on multiple objectives, so the parents whose offspring are most likely to advance the multiobjective search are selected more often.

We use two approaches to achieve this, depending on the kind of multiobjective map we use. In the case of multi map, we alternate between which metric is used (depending on which update mode is used) for weighting the individuals in the archive. In the case of non dominated map, we do a non dominated sort on the whole archive, and use the ranks of that sort to weigh the individuals. This is different from how MOME does selection with crowding exploration [54], they select a random cell first than do the nondominated sort, compared to doing the nondominated sort on the whole archive. The pseudocode for the parent selection algorithm is available in Listing 5.6.

```
def rank_based_selection(num_parent_candidates,agressiveness=1.0):
    # agressiveness decide how agressively we want to select higher ranking individuals
    over lower ranking ones.
    # 1 is normal rank based probability, 0 is uniform,
    # The higher it is the more agressive the selection is.
    p = np.array(list(range(num_parent_candidates))) + 1.0
    p = p ** agressiveness
    p = p / np.sum(p)
    return np.random.choice(num_parent_candidates, p=p)

def select_from_map(evo_map_archive,metric):
    if config["use_biased_selection"]:
        metrics = [eval[metric] for theta,eval in evo_map_archive]
        sorted_indicies = np.argsort(metrics)
        winner_i = rank_based_selection(len(evo_map_archive))
        return evo_map_archive[sorted_indicies[winner_i]]
    else:
        return np.random.choice(evo_map_archive)

def select_from_nondominated_map(evo_map_archive,metrics):
    all_elites = []
    for cell in evo_map_archive:
        for individual in cell:
            all_elites.append(individual)
    if config["use_biased_selection"]:
        metrics = config["nd_map_archiving_metrics"]
        sorted_indicies = nd_sort_individuals(all_elites,metrics)
        winner_i = rank_based_selection(len(all_elites))
        return all_elites[sorted_indicies[winner_i]]
    else:
        return np.random.choice(all_elites)

def parent_selection(evo_map_archive,parameter_update_type):
    if config["archive_mode"] == "single_map":
        return select_from_map(evo_map_archive,config["single_map_archiving_metric"])

    if config["archive_mode"] == "multi_map":
        # select parent based on the used parameter_update_type.
        matching_metric = {
```

```
            "es" : "adaptability",
            "evolvability_es" : "evolvability",
            "novelty_es" : "innovativeness",
        }[parameter_update_type]
        matching_metric_i = config["multi_map_archiving_metrics"].index(
    parameter_update_type)
        return select_from_map(evo_map_archive[matching_metric_i],matching_metric)

    if config["archive_mode"] == "non_dominated_map":
        return select_from_nondominated_map(evo_map_archive,config["
    nd_map_archiving_metrics"])
```

Listing 5.6: Parent selection algorithm for Evolvability Map Elites

The advantage of doing the nondominated sorting selection is that it tends to select individuals with various tradeoffs between the objectives, while simply alternating between the selection metrics only tends to select individuals who are good at one metric irrespective of how they perform in other metrics.

In this work, we use alternating metrics for multi map map elites, because it already ignores tradeoffs between objectives, and only archives the best for each metric separately. However, for nodominated sorted map we use nondominated sorting parent selection. Because the map insertion decision already cares about the tradeoff between objectives, it makes sense to also focus on the tradeoffs for parent selection.

We note however that this decision is not that straightforward. It might be beneficial to mix these approaches, to introduce some pressure for tradeoff, or some pressure for maximizing individual metrics. This is something which we plan to investigate in the future.

5.3.7 Lineage lengths

The strength of map elites comes from being a divergent algorithm. It can explore the search space in ways that an objective-based algorithm is not able to. However, because being a divergent algorithm, it has a lot of branches in its phylogenic tree, causing the depth of these branches to be relatively low. There are only a relatively few consecutive updates an individual typically receives since the randomly initialized root individual.

The divergent nature of map elites was not an issue for the original map elites. With normal map elites, we create a child individual after every evaluation (or a couple of evaluations in case of stochastic tasks). The high branching was compensated by the

number of children produced, so the depth of the phylogenic tree remained reasonable. The situation changed however with the introduction of ES map elites. For Map Elites ES, we need to evaluate a whole population just to create one child, where a commonly used population size is 10000.

This kind of divergent search combined with orders of magnitude fewer children generated can create issues. One issue is that with fewer children, it is harder to fill up the map. This can be observed in the original result for Map Elites ES [91] where the map is orders of magnitude less populated for Map Elites ES compared to normal map elites. Another issue is the shallowness of the phylogenic tree. Sometimes we see with ES that we need hundreds of consecutive gradient updates to reach a breakthrough[32]. With the branching nature of Map Elites ES, we might never get a hundred consecutive steps on any branch of the phylogenic tree.

In our case, we worsened the situation further with the introduction of multi objective map elites. Now the size of the map is even larger, causing even more pronounced branching (from multi map with 3 metrics, we have 3 times the map size, with nd map with 5 elite per cell we have five times the map size).

One way to solve this issue is to reduce the resolution of the map, making it smaller. But this would sacrifice the quality of the output. Luckily with the way we separated elite performers and elite evolvers into separate maps (5.3.4), we can reduce the size of only the evolver map. This reduces branching since we only select parents from the elite evolver map. This way we can reduce branching, while still being able to produce a high-resolution behaviour map. For both of our tasks, we quadrupled the cell size causing a 16-times reduction in map size for the evolver map compared to the performer map (from 32 x 32 (1024 cells) to 8 x 8 (64 cells)).

5.3.8 Innovation recalculation

One of the evolvability metrics we use is innovativeness, which is the ability to generate novelty (expected novelty of offspring). There are some special considerations when working with innovativeness. This is because the meaning of innovation continuously changes, as the archive of already seen behaviours gets larger. What was previously considered in-

novative we may no longer see as innovation as time passes. The only fair way to compare the innovativeness of two individuals is to compare them against the same archive.

We always need to recalculate innovation when we want to compare the innovation of 2 individuals. Recalculating innovation is easy because we store the results (fitness and behaviour) of the sampled offspring of every individual. No extra evaluations are required. Storing this data does not consume a lot of memory, because we only need to store the results (behaviours of the population), not the parameters. (Although storing the parameters is also cheap thanks to using a random table indicies[32]). We found that this recalculation of innovation does not add a significant cost to the algorithm, especially since it can be done efficiently on the GPU.

5.3.9 Accelerated physics simulation

In section 5.3.1. we explained how we are able to reuse the same computation multiple times, which provides us with an efficient way to optimize for evolvability. However efficient it may be, we are still using ES, which requires a lot of computation. In this chapter, for a single experiment, we do 50 million evaluations. (10k population size times and 5k generations). To evaluate the algorithm we use the benchmark task of robotics simulations, like many previous papers in the area.

Previously ES algorithms were run on CPU clusters because the largest bottleneck was running the simulations, which run on the CPU, so the best way to speed up the simulation was to add more CPU cores. Luckily the state of GPU programming improved a lot in recent years. Now people can utilize the GPU with much less effort, for all kinds of computation. One big step in this direction is the creation of XLA, which we discussed in the background (chapter 3.). XLA uses Just In Time (JIT) compilation to turn linear algebra graphs into device code. The JAX [83] library utilizes XLA to provide the user with a numpy-like API to create these linear algebra graphs. With the accessibility of this kind of simple API, it became easy to build complex programs on the GPU, like the physics engine Brax [85].

Brax allows us to run many similar physics elements concurrently. This is not a general physics engine, it only excels for the kind of problem, where we need to run the

exact same setting many times, like in RL or evolutionary computation. (Brax also has some other interesting properties, for example, it allows for the physics to be differentiated). With Brax, we can run tens of thousands of environments on the GPU. With Brax, running these experiments became much more accessible (GPU-s are more widespread than CPU clusters), faster, cheaper and consume much less energy. It is truly a game changer for this kind of research.

5.3.10 Accelerated algorithm

The reason we discuss the accelerated physics simulation is that it has implications for algorithm design. With such speedup of the largest chunk of computation: the evaluations, there are new bottlenecks. Previously insignificant components of the algorithm can now have large impacts on the runtime.

To keep up with the pace, we moved many other parts of the algorithm to the GPU as well. This includes the model forward pass, the novelty and innovation calculations and the gradient calculation. When making the decision about what to calculate on the GPU and what on the CPU, we have to take into consideration where our data is located. Because we do the evaluations on the GPU, it makes sense to do the gradient calculation on the GPU as well, since all the data necessary for those calculations are on the GPU already. By doing it on the GPU, we do not just save on the actual computation runtime, but also save the time to copy all that data to the CPU. Another bottleneck which appeared is the nondominated sorting, which is used in many places in the algorithm, (combined ES updates, map insertion decision, parent selection). Nondominated sorting has 2 components which take up most of the computation. One is the domination matrix calculation, which we also moved to the GPU when the input size is large. The other is the front calculations, which include several steps [89]. We kept this part on the CPU, but we applied JIT compilation with Numba [105] to make it faster.

5.4 Experiment

Our original goal in this chapter is to create a map elites version which optimizes for evolvability. To achieve this, we introduced several components for map elites in the previous section. These components raise many questions.

5.4.1 Questions

Question 1: is evolvability beneficial to map elites in any of its objectives?

The most important question we want to answer is whether it is beneficial to add evolvability to map elites. When we say beneficial we mean does the performance of the algorithm increases in any of its goals: quality, diversity, evolvability, and a combination of these. We will discuss how we quantify and compare performance in each of these metrics in the next section.

Question 2: What form of evolvability pressure is beneficial for map elites? (evolvability update, evolvability selection or a combination of the two).

We added the possibility to do optimization for evolvability, and selection for evolvability (via the insertion decision). It is not clear what is the best way to introduce pressure for evolvability in a way which enables the synergies with the other objectives instead of hindering it. Too much pressure for evolvability might be detrimental because we miss the stepping stones which can be found looking for other objectives besides evolvability (quality, diversity). The question is, is it better to do both evolvability optimization and selection or only do one of them?

Question 3: Is it better to combine multiple metrics for selection with an AND or OR relationship?

When doing the archiving decision with multiple metrics, we proposed two mechanisms. The multi map version inserts individuals in the map if they are the elite with respect to any metrics. This creates an OR relationship with the metrics, an individual is inserted if it is high fitness or high evolvability or high innovativeness. The non dominated map version inserts individuals into the map if the individual is not dominated by the other elites already present in the cell. This creates an AND relationship between the metrics. The individual is inserted if it has a high fitness and high evolvability and high innovativeness. It is not clear which method leads to higher synergy between the objectives.

Question 4: Is it better to select for expected fitness than evaluation fitness with Map Elites ES?

Previous versions of Map Elites ES made the map insertion decision based on evaluation fitness. However, the objective of the ES update is the expected fitness of your offspring. This created a situation where the optimization and selection are not working towards the same goal. With the one step lookahead computation we do, we are able to select for expected fitness now. We hypothesise that selecting for the same thing which the update optimizes for will increase the performance of map elites.

5.4.2 Algorithm Variants

In order to attempt to answer these questions, we created several variants of the algorithm. Table 5.2. describes which variant has which component enabled.

Algo name	**ES update objective**	**Map insertion decision metric**
ME-ND_full	adaptive, diverse, innovative	adaptiveness **and** evolvability **and** innovativeness
ME-MM_full	adaptive, diverse, innovative	adaptiveness **or** evolvability **or** innovativeness
ME-ND_explore_exploit	adaptive, innovative	adaptiveness **and** evolvability **and** innovativeness
ME-MM_explore_exploit	adaptive, innovative	adaptiveness **or** evolvability **or** innovativeness
ME-ND_exploit	adaptive	adaptiveness **and** evolvability **and** innovativeness
ME-MM_exploit	adaptive	adaptiveness **or** evolvability **or** innovativeness
ME-no_selection	adaptive, diverse, innovative	adaptiveness
ME-no_selection_eval	adaptive, diverse, innovative	fitness
ME-explore_exploit	adaptive, innovative	fitness
ME-exploit	adaptive	fitness

Table 5.2: Algorithm variants used in the experiments. There are 3 kinds of ES updates, one which maximizes expected fitness of offspring (adaptive), expected diversity of offspring (diverse) and expected novelty of offspring (innovative). The map insertion decision metric determines the metric which is used to select individuals which are inserted in the map.

The baseline algorithm we use is the Map Elites ES variants introduced in [91]. The ME-exploit is baseline where only fitness ES update is used. With ME-explore-exploit there are two kinds of updates, fitness and innovativeness. For both benchmarks the insertion decision is based on evaluation fitness (same as for all previous map elites variants).

To answer questions 1 and 2, is evolvability beneficial, and what kind of evolvability pressure works well, we created several algorithm variants. There is a variant with evolvability update, but no evolvability selection. There are also variants for evolvability selection but no evolvability update. For the no selection case, we created 2 variants, an exploit variant with only fitness based ES update, and an explore_exploit variant with both fitness and innovativeness updates. Finally, we also created the full variants which use both evolvability selection and evolvability update.

To answer question 3, for algorithms which use evolvability selection we created two variants, one with nondominated map (ND) and one with multi map (MM). The difference between these is that ND uses an AND relationship between the metric for selection, while MM uses an OR relationship.

To answer question 4 for the no evolvability selection algorithm we created a version where we select for evaluation fitness (like all previous versions of map elites) called ME-no_selection_eval and a version where we select for expected fitness (adaptiveness) which is called ME-no_selection.

For comparison, we also did runs with the plain ES algorithm and its variants (see table 5.3.). We used the fitness based ES, and the two variants introduced in the previous chapter 4 evolvability ES (E-ES) and quality-evolvability ES (QE-ES). We also introduced a new ES variant called Quality Evolvability Diversity ES (QED-ES), where instead of only using fitness and evolvability updates, we also do innovativeness updates. Since in the case of ES, the output of the algorithm is a single individual, not an archive of individuals like in map elites, diversity is not defined for ES. For this reason, the ES baselines were only used for comparison for the fitness and evolvability metrics.

ES variant name	Objective
ES	Fitness
Evolvability ES (E-ES)	Evolvability
Quality Evolvability ES (QE-ES)	Fitness, Evolvability
Quality Evolvability Diversity ES (QED-ES)	Fitness, Evolvability, Diversity

Table 5.3: Plain ES variants used as baselines. E-ES and QE-ES were discussed in the previous chapter. QED-ES was introduced in this chapter, where instead of only using the 2 kinds of objectives in the ES update, we use all three (expected fitness, expected diversity, expected novelty).

5.4.3 Tasks

We evaluated our method on the robotics locomotion tasks Ant and Humanoid. Compared to the previous chapter, where the algorithm was tested on the implementation using the PyBullet physics engine, for these experiments, we use the Brax physics engine implementations of the tasks. The reason for switching to Brax is to be able to run the physics on the GPU.

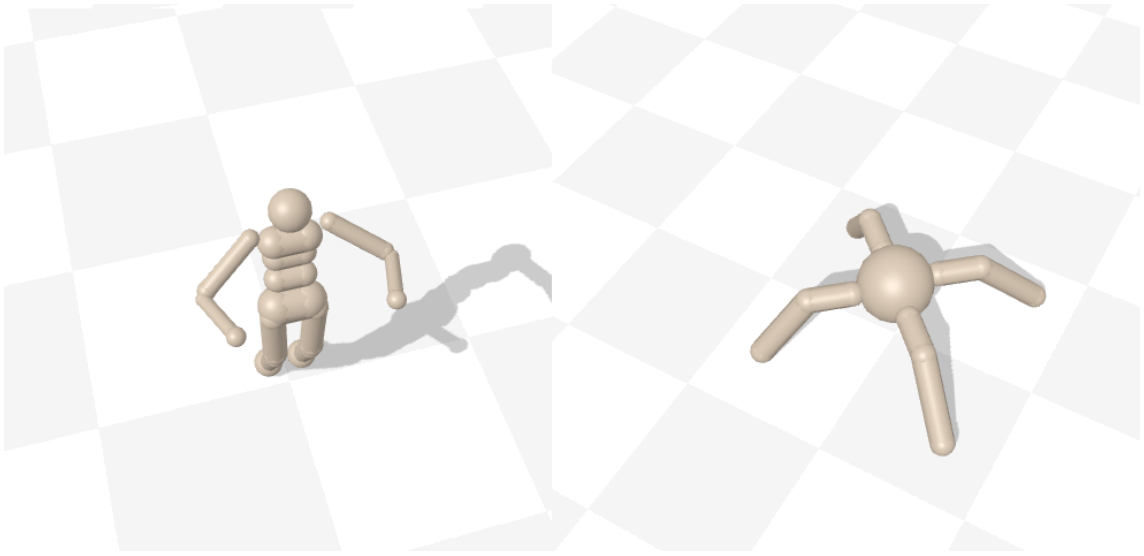


Figure 5.2: The robots from the brax humanoid (left) and ant (right) environments. Fitness is the distance walked by these robots, while the behaviour descriptors are the x and y coordinates of the final position of the robots at the end of the simulation.

The simulation is mostly identical to the PyBullet version which we used in the previous chapter, the robots have the same configurations, dimensions, inputs and outputs. For the details of the environment see section 4.5. There are some differences in how the physics is calculated internally since Brax needs to use simplified formulas so the update can be parallelized efficiently. These differences, however, do not seem to be important

from the perspective of the learning algorithms, since the learned behaviours do not differ significantly compared to the other implementations (MuJuCo and PyBullet) [85].

We used the ant and humanoid environments. For both environments, the fitness is calculated as the distance between the starting position and the final position of the robot at the end of the simulation. The behaviour descriptor is the x and y coordinate of the final position of the robot. The behaviour grid limits were 50 meters from the origin in every direction.

5.4.4 Training Details

The controller for these tasks was a two hidden layer neural network with a hidden size of 64 and relu activation function. Ant has 87 inputs and 8 outputs, while the Humanoid has 240 inputs and 17 outputs. This leads to a network size of a little more than 10 thousand for Ant and 20 thousand for Humanoid. The performer map elite grid size is 32,32 (1024 cells) while the evolver map grid size is 8,8 (64 cells). An episode consists of 250 timesteps. We use an ES population size of 10000. ES σ was 0.02. The Adam optimizer was used with a learning rate of 0.01 and L2 decay of 0.005. We pick a new parent and update mode every 10 generations, like the original Map Elites ES [91]. The training was run for 5000 generations. For the plain ES runs we used identical hyperparameters as for the map elite variants (learning rate, ES σ, L2 decay) however we only run them for 500 generations, as ES converges much faster than Map Elites ES. We used rank proportional parent selection. The metric used to determine rank was decided as described in the parent selection section 5.3.6. We used observation normalization as in the original ES paper [32]. We found that observation normalization plays a critical role, without it the performance drops drastically.

5.5 Results

We run each 10 variants of map elites on the 2 environments for 10 repeated runs each, for a total of 200 runs. For the ES baselines, we have 4 variants with 2 environments with 10 repeated runs each for a total of 80 runs.

The source code used to produce these results is available at `https://github.com/adam-katona/evolvability_map_elites`.

The kind of algorithms we are creating here have multiple objectives, to evaluate them properly we need multiple metrics. We have 5 metrics which are discussed in the next section. The summary of the results is presented in 5 tables (from 5.4 to 5.8), one for each metric.

5.5.1 Metrics

We have the three metrics we discussed earlier in this chapter, quality, diversity and evolvability. Quality is measured as the evaluation fitness of the best individual in the map. Diversity is measured as the ratio of nonempty cells in the map. Evolvability is measured as the evolvability (entropy of the offspring distribution) of the most evolvable individual in the map.

Looking at our objectives alone is not enough to properly compare the algorithm variants. The reason we want to do multi objective search is because we hope that there are synergistic relationship between the objectives. By looking for one objective we can find stepping stones for the other.

If we have a hundred different ways to fail a task, we might have diversity, but that might not bring us closer to a good solution. But if we have a hundred different ways to make some progress, that is much more likely to help us find new and innovative solutions which might lead us to high performance eventually.

Similarly with evolvability, if we have one individual which has a diverse set of offspring, that is great, but if we have a 100 different individual which all have diverse offspring, we are maybe more likely to find some innovative new solution.

To quantify these ideas we use the metrics quality diversity (QD) and evolvability diversity (ED). Quality diversity was introduced in the original map elites paper [91], it is calculated by summing the fitness of all the elites in the map. We defined evolvability diversity to be calculated in a similar fashion, we sum the evolvability of all the elites in the map.

algo_name	ant	humanoid
ES	57.9±2.6	7.9±0.9
E-ES	12.6±4.0	3.7±0.2
QE-ES	28.1±2.7	5.1±1.0
QDE-ES	32.1±8.7	3.2±0.3
ME_exploit	69.8±3.6	9.0±3.1
ME_explore_exploit	40.8±5.5	7.0±2.3
ME_no_selection_eval	26.4±8.9	6.8±1.7
ME_no_selection	28.7±5.4	7.3±1.1
ME-MM_exploit	38.6±7.8	6.9±1.3
ME-ND_exploit	39.1±7.1	7.5±2.3
ME-MM_explore_exploit	67.4±5.3	9.8±2.0
ME-ND_explore_exploit	68.4±11.1	9.5±3.1
ME-MM_full	27.2±8.6	7.1±1.3
ME-ND_full	30.6±3.9	8.5±1.6

Table 5.4: Quality scores for all algorithm variants. Quality is calculated as the best evaluation fitness in the map, or the final evaluation fitness in the case of the plain ES algorithms. The reported values are the mean and standard deviation from ten repeated evaluations.

5.5.2 General observations

The results suggest that there is no one method which is best. When we say one algorithm is better than another, we mean that for most or all of the metrics it is better. For different environments and metrics, different variants work best. Even when considering the same metric, for the two environments often different algorithm variant performs best. This suggests that what is the best way to do map elites is very much task dependent. We can also obverse that in some cases there seems to be a tradeoff between the objectives, making a change which leads to better performance in another objective sometimes result in worst performance in the original objective. However, we can also observe cases where making a change which adds pressure towards another objective leads to better performance in both objectives. Whether these relationships are synergistic or are a tradeoff seems to be task dependent as well. Even though there seems to be no clear winner, the results allow us to draw some conclusions with respect to the four questions we presented in the previous section.

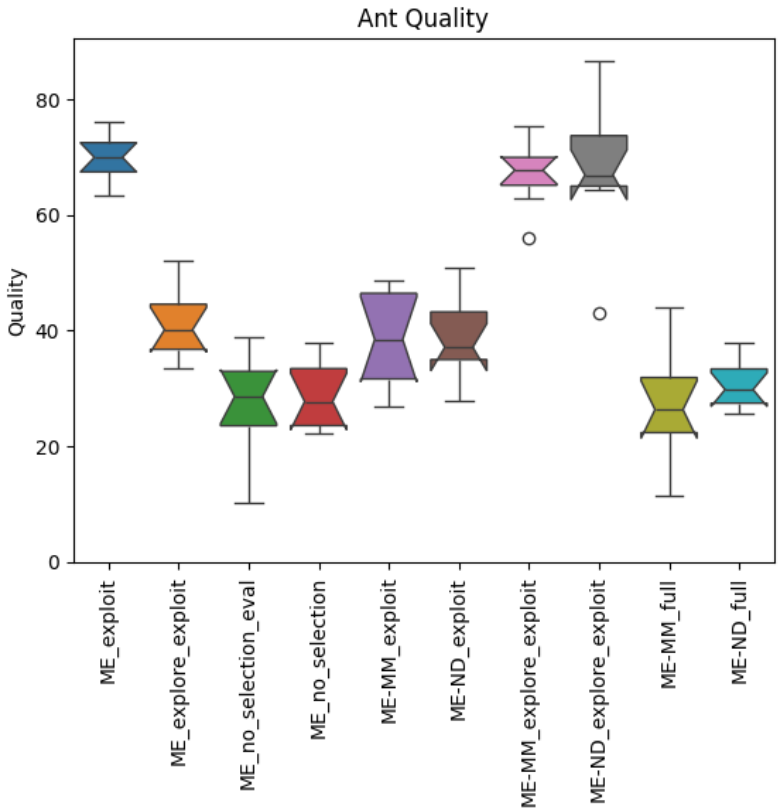


Figure 5.3: Quality scores for each variant of Evolvability Map Elites on the ant task

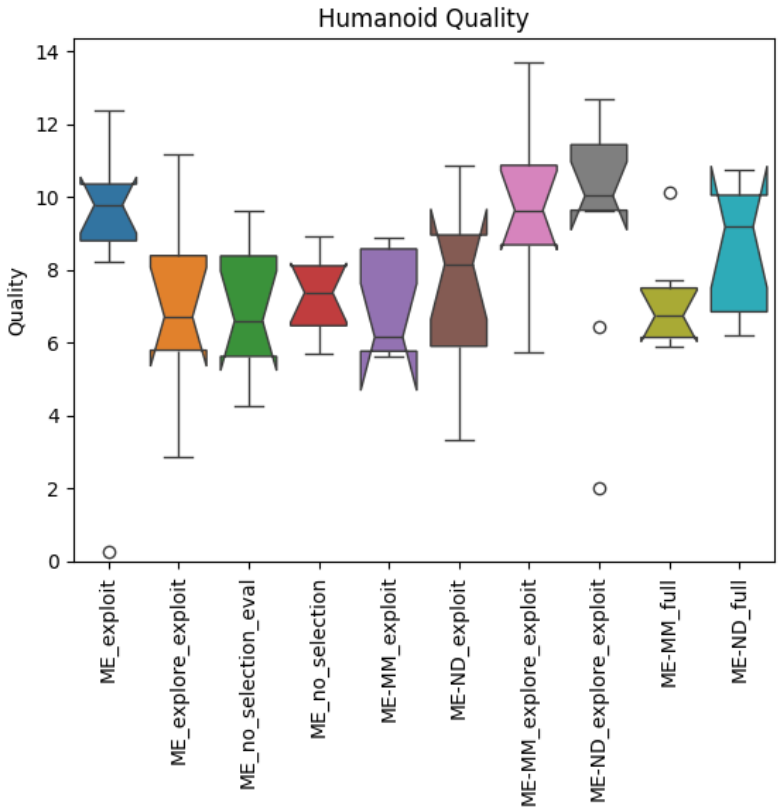


Figure 5.4: Quality scores for each variant of Evolvability Map Elites on the humanoid task

algo_name	ant	humanoid
ME_exploit	0.093±0.017	0.007±0.002
ME_explore_exploit	0.220±0.040	0.010±0.004
ME_no_selection_eval	0.113±0.047	0.009±0.003
ME_no_selection	0.123±0.030	0.012±0.003
ME-MM_exploit	0.225±0.054	0.009±0.002
ME-ND_exploit	0.221±0.041	0.010±0.004
ME-MM_explore_exploit	0.088±0.016	0.007±0.002
ME-ND_explore_exploit	0.080±0.020	0.007±0.002
ME-MM_full	0.113±0.044	0.011±0.003
ME-ND_full	0.140±0.015	0.013±0.004

Table 5.5: Diversity score for all algorithm variants. Diversity is calculated as the ratio of non empty cells in the map compared to the size of the map. The reported values are the mean and standard deviation from ten repeated evaluations.

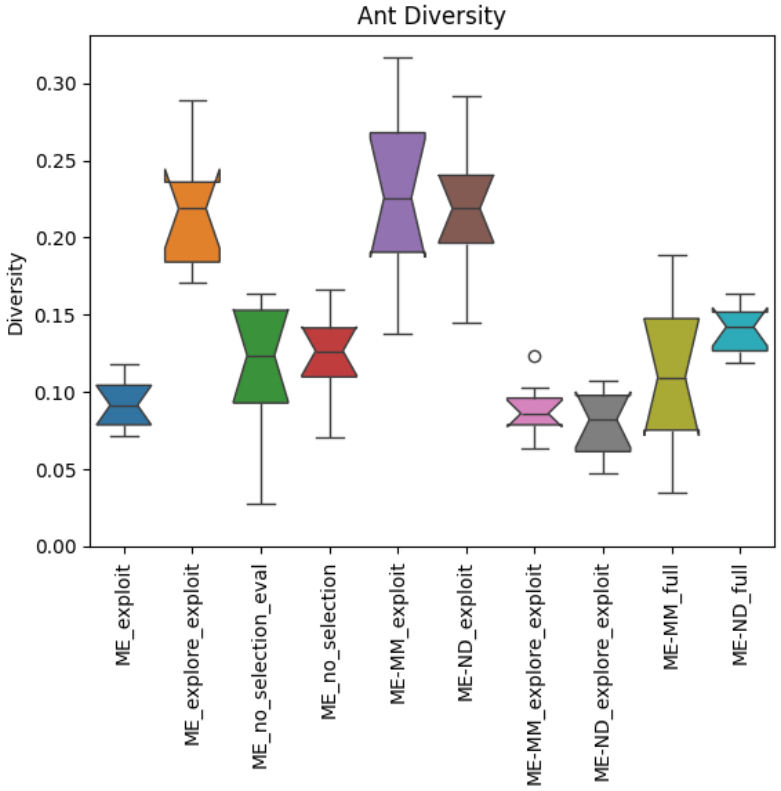


Figure 5.5: Diversity scores for each variant of Evolvability Map Elites on the ant task

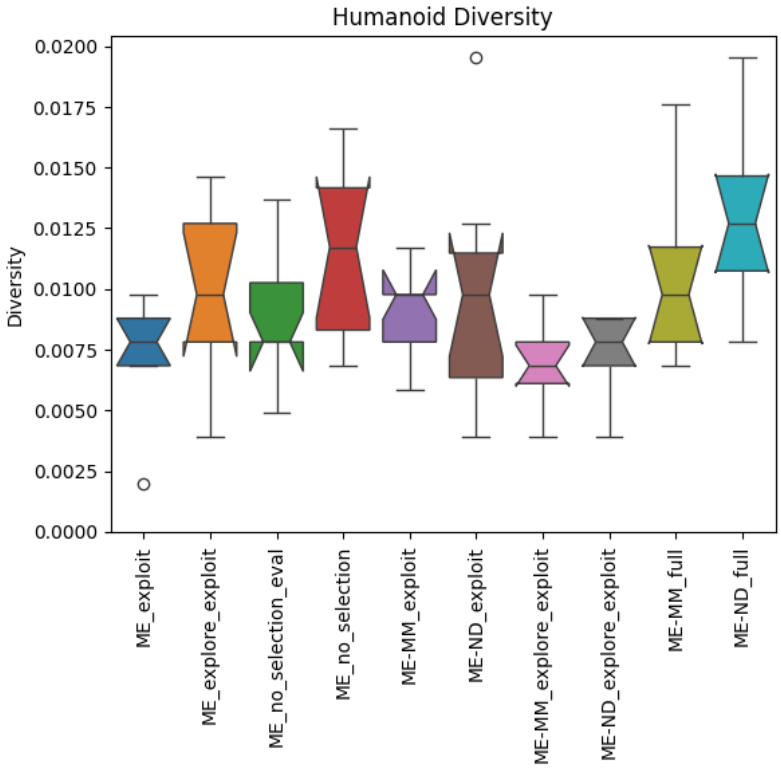


Figure 5.6: Diversity scores for each variant of Evolvability Map Elites on the humanoid task

algo_name	**ant**	**humanoid**
ES	6.6±0.1	4.7±0.1
E-ES	6.7±0.1	4.8±0.0
QE-ES	6.7±0.0	4.9±0.2
QDE-ES	6.8±0.0	4.5±0.0
ME_exploit	8.3±0.1	4.5±1.5
ME_explore_exploit	8.0±0.1	4.9±0.5
ME_no_selection_eval	8.1±0.4	5.3±0.4
ME_no_selection	8.3±0.0	5.3±0.3
ME-MM_exploit	8.0±0.1	4.9±0.3
ME-ND_exploit	8.0±0.1	5.1±0.4
ME-MM_explore_exploit	8.3±0.1	5.0±0.3
ME-ND_explore_exploit	8.2±0.2	4.7±0.7
ME-MM_full	8.2±0.3	5.5±0.3
ME-ND_full	8.4±0.1	5.7±0.3

Table 5.6: Evolvability scores for all algorithm variants. Evolvability is calculated as the entropy of the offspring distribution of the individual. For map elites entropy score is the highest entropy of any elites in the map. The reported values are the mean and standard deviation from ten repeated evaluations.

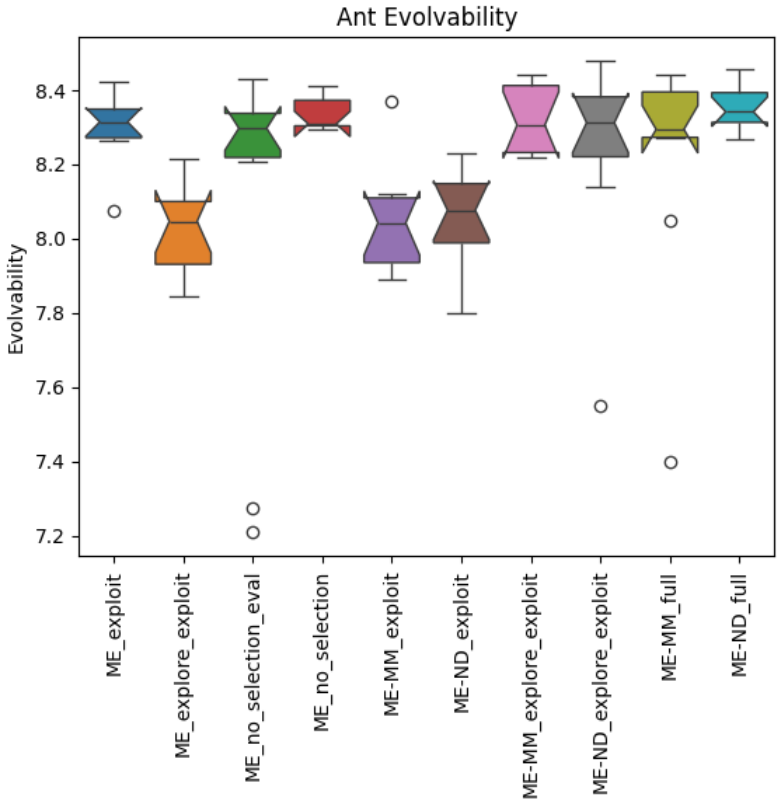


Figure 5.7: Evolvability scores for each variant of Evolvability Map Elites on the ant task

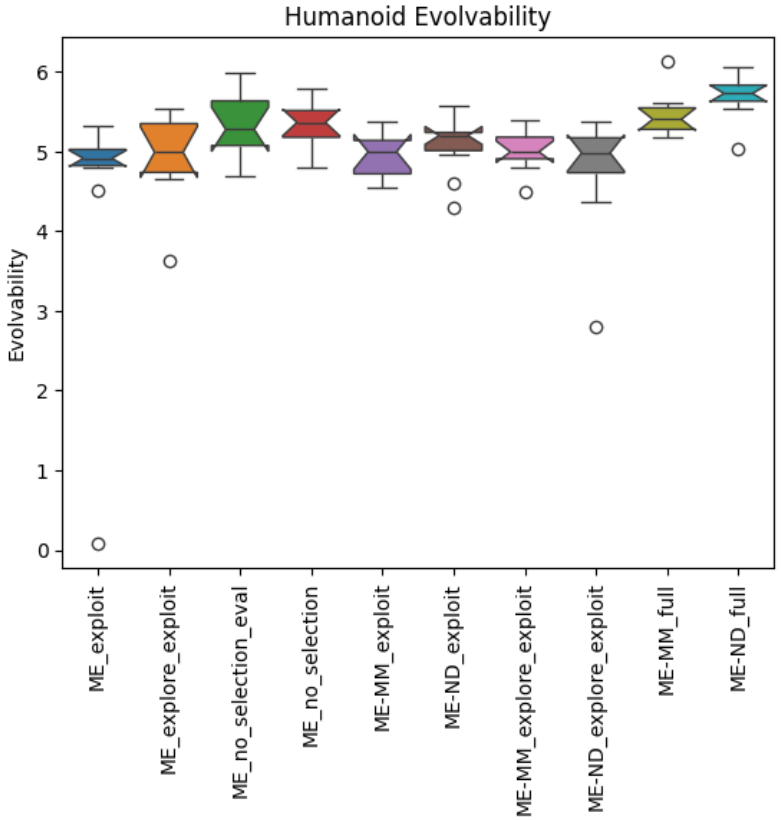


Figure 5.8: Evolvability scores for each variant of Evolvability Map Elites on the humanoid task

algo_name	ant	humanoid
ME_exploit	4183.0±935.8	46.6±17.9
ME_explore_exploit	5237.9±1408.3	49.5±27.6
ME_no_selection_eval	2180.0±1139.0	44.1±22.7
ME_no_selection	2381.5±760.6	65.2±30.6
ME-MM_exploit	5288.7±1810.5	42.1±12.8
ME-ND_exploit	5298.0±1610.6	54.8±40.2
ME-MM_explore_exploit	3800.2±881.9	48.4±20.6
ME-ND_explore_exploit	3519.8±1103.3	46.8±20.3
ME-MM_full	2252.7±1218.7	56.0±30.1
ME-ND_full	2777.9±377.5	83.5±38.7

Table 5.7: Quality Diversity (QD) scores for all algorithm variants. QD is calculated by summing the evaluation fitness of the elites in the map. The reported values are the mean and standard deviation from ten repeated evaluations.

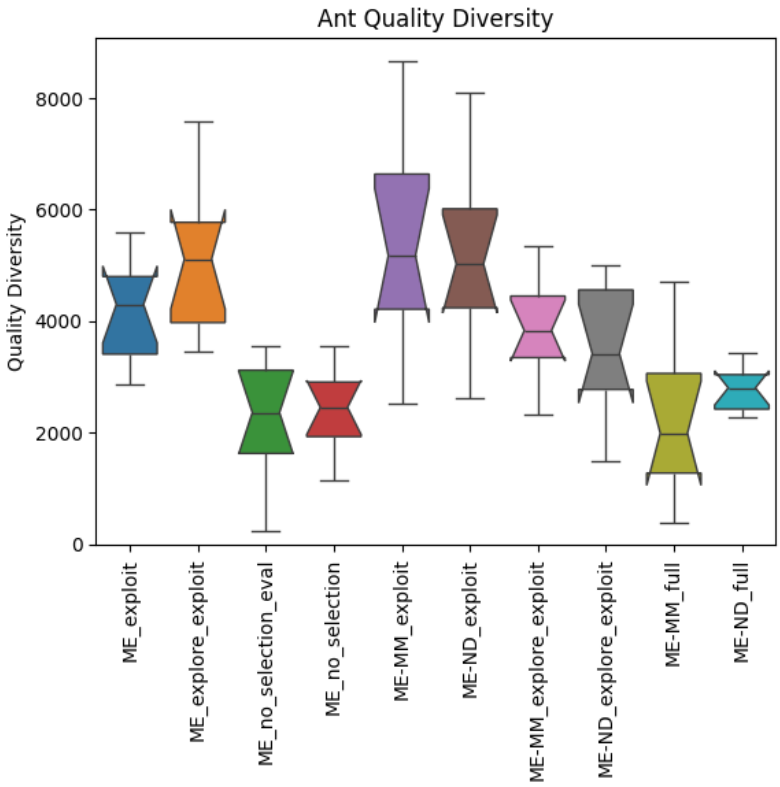


Figure 5.9: Quality Diversity scores for each variant of Evolvability Map Elites on the ant task

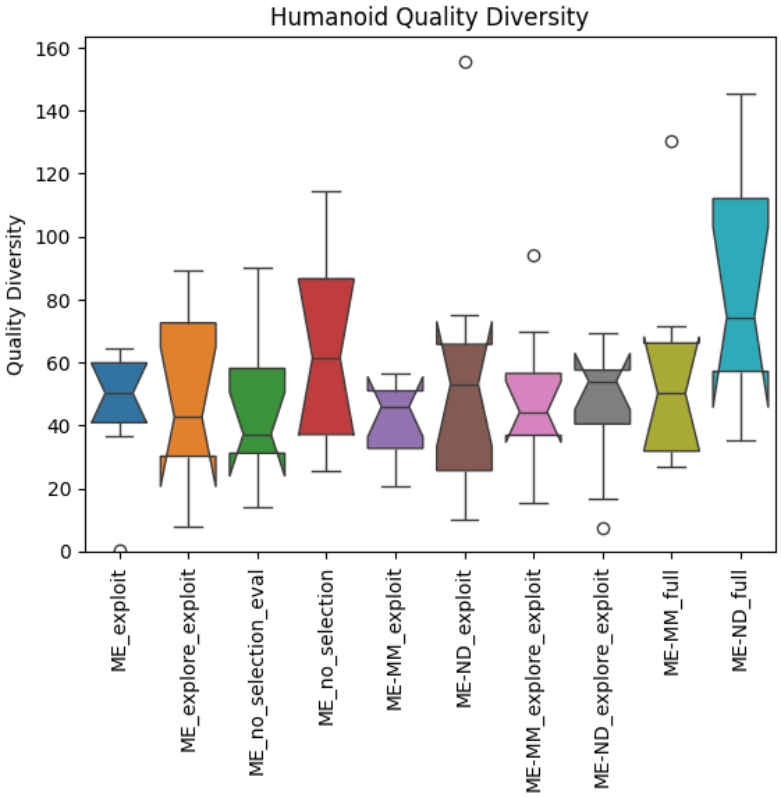


Figure 5.10: Quality Diversity scores for each variant of Evolvability Map Elites on the humanoid task

algo_name	**ant**	**humanoid**
ME_exploit	89.5±12.8	8.5±3.1
ME_explore_exploit	183.4±30.8	15.1±2.5
ME_no_selection_eval	102.8±47.2	14.1±1.8
ME_no_selection	114.3±25.3	15.2±2.2
ME-MM_exploit	190.5±40.8	14.7±2.1
ME-ND_exploit	190.2±42.2	15.1±1.6
ME-MM_explore_exploit	90.4±16.4	9.3±2.8
ME-ND_explore_exploit	88.4±14.0	9.2±1.6
ME-MM_full	109.2±44.4	17.9±2.0
ME-ND_full	134.2±15.6	18.2±1.6

Table 5.8: Evolvability Diversity (ED) scores for all algorithm variants. ED is calculated by summing the evolvability of all the elites in the map. The reported values are the mean and standard deviation from ten repeated evaluations.

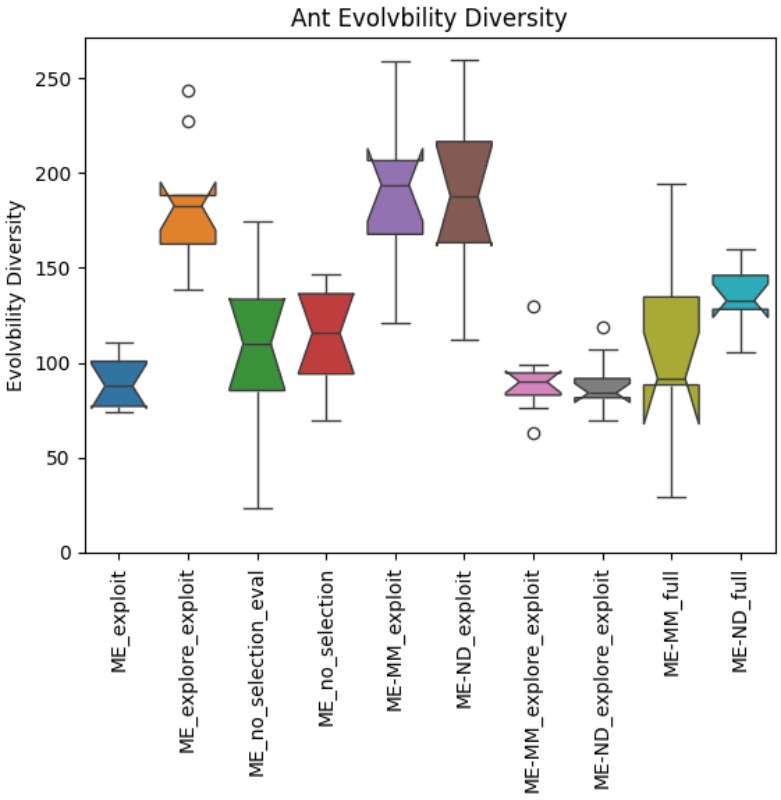


Figure 5.11: Evolvbility Diversity scores for each variant of Evolvability Map Elites on the ant task

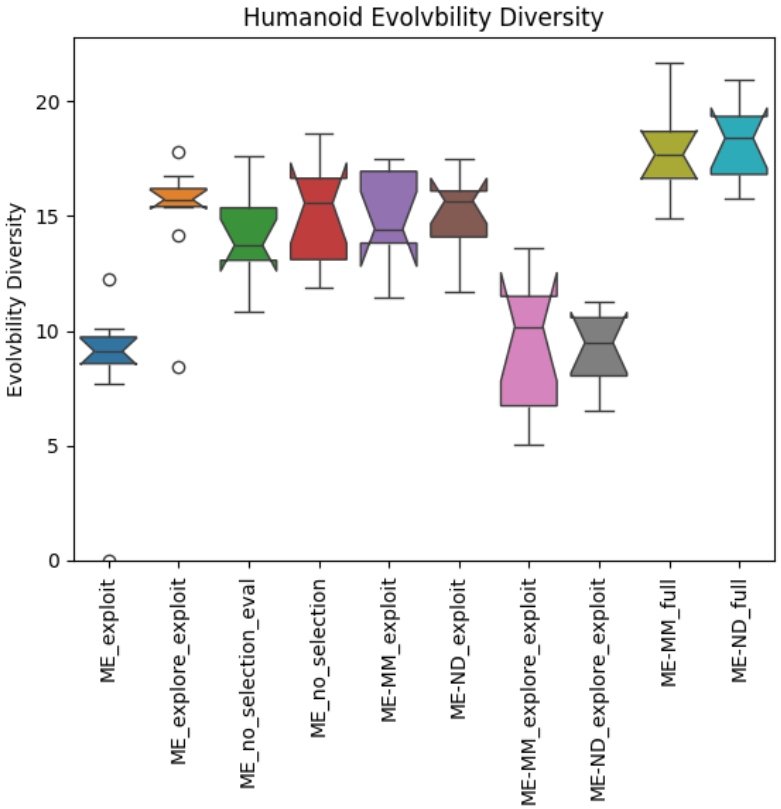


Figure 5.12: Evolvbility Diversity scores for each variant of Evolvability Map Elites on the humanoid task

Group	Algorithm variants
full (evolvability selection and update)	ME-MM_full ME-ND_full
only evolvability update	ME_no_selection ME_no_selection_eval
only evolvability selection	ME-MM_exploit ME-ND_exploit ME-MM_explore_exploit ME-ND_explore_exploit
no evolvability	ME_exploit ME_explore_exploit

Table 5.9: We grouped the algorithm variant into four groups based on what kind of evolvability component they contain. For the comparison between these groups, we selected the best performing variant for each task and metric

5.5.3 Evolvability in map elites

The first 2 questions we asked are: is it beneficial to add evolvability to map elites, and what kind of evolvability pressure works best? To answer the second question we created 4 groups of algorithm variants. These are summarized in table Tab 5.9. The groups are based on what kind of evolvability component is utilized by the variants. To answer the first question, we further reduced the algorithms into two groups, one with any evolvability component, and the other without any evolvability component. For the comparison between these groups, we took the best for each group.

Question 1: is evolvability beneficial to map elites in any of its objectives? To test this we have 10 test cases, which come from 5 metrics and 2 environments. We found that map elites variants which have some kind of evolvability component achieved highest maximum score out of all the runs compared to variants without evolvability for 9 out of the 10 cases (Tab 5.10).

Question 2: What form of evolvability pressure is beneficial for map elites? For this we have the same 10 test cases, however, we have 4 groups now: no evolvability, evolvability selection, evolvability update, or both. We found that out of 10 cases in 1 no evolvability , in 4 cases only evolvability selection , and in 5 cases using both evolvability selection and update achieved highest maximum score out of all the runs. Evolvability update was the best in none of the cases and was the worst in many cases. It seems to be the case that while evolvability selection is beneficial on its own, evolvability update is

metric, environment	map elites (no evolvability variants)	evolvability map elites (any variant)
quality, ant	69.78	68.37
quality, humanoid	8.98	9.77
diversity, ant	0.220	0.225
diversity, humanoid	0.0095	0.0131
evolvability, ant	8.29	8.35
evolvability, humanoid	4.91	5.68
QD, ant	5237.94	5297.97
QD, humanoid	49.50	83.52
ED, ant	183.40	190.54
ED, humanoid	15.09	18.19

Table 5.10: This table summarizes data for answering question 1: Is adding evolvability to map elites beneficial? We found that map elites variants which have some kind of evolvability outperform variants without evolvability for 9 out of 10 cases. The table shows the scores of the best performing variant for each metric and environment.

task_name	no evolvability	only_selection	only_update	full
quality, ant	69.78	68.37	28.73	30.63
quality, humanoid	8.98	9.77	7.28	8.53
diversity, ant	0.220	0.225	0.122	0.139
diversity, humanoid	0.0095	0.0096	0.0116	0.0131
entropy, ant	8.29	8.32	8.33	8.35
entropy, humanoid	4.91	5.09	5.34	5.68
QD, ant	5237.94	5297.97	2381.47	2777.87
QD, humanoid	49.50	54.77	65.19	83.52
ED, ant	183.40	190.54	114.28	134.17
ED, humanoid	15.09	15.10	15.23	18.19

Table 5.11: This table summarizes data for answering question 2: What form of evolvability pressure is beneficial for map elites? We found that out of 10 cases in 1 no evolvability performed best, in 4 cases only evolvability selection performed best, and in 5 cases using both evolvability selection and update performed best.

not beneficial on its own. However when combined, and both of them are enabled, we get the best performance. We note that this is a very limited number of cases, and to get a more general picture of which mode works best, we need to test in more environments.

5.5.4 Multi map vs ND map

Question 3: Is it better to combine multiple metrics for selection with an AND or OR relationship? To answer this question, we have 3 algorithms which we tested with both nondominated map (AND relationship) and multi map (OR relationship). With 5 metrics and 2 environments, this creates 30 test cases. We found that the decision to use

either nondominated map or multimap does not seem to influence the results in these two environments. With applying the Bonferroni correction none of the cases were statistically significantly different (Mann-Whitney U Test).

5.5.5 Selecting for ES objective vs eval fitness

Question 4: Is it better to select for expected fitness than evaluation fitness with Map Elites ES? To answer this question we compare the algorithm variant ME_no_selection and ME_no_selection_eval. The difference between these is whether expected fitness or evaluation fitness is used as the metric for insertion decision. We tested with 5 metrics and 2 environments leading to a total of 10 tests. We found that with applying the Bonferroni correction there is no statistically significant differences in all 10 cases (Mann-Whitney U Test).

5.6 Conclusion

In this chapter, we created variants of map elites with various evolvability pressures. We can conclude that adding evolvability to map elites did result in improvements in the objectives of map elites for some of the cases. However, these improvements are not consistent (not for all metrics / tasks) and not transformational.

5.6.1 Improving the consistency of results

The observed results did not show consistency, with different environments, different algorithm variants performed best. We can even observe cases where one variant performed best for an environment, while the same variant was one of the worst for a different environment. To overcome this task dependence, we need an algorithm which automatically discovers which variant is best for each task. Such an algorithm would dynamically turn on and off different components of the algorithm to try out different balances in the pressures for each objective.

5.6.2 A new innovativeness metric

Another possible path to improvement is to use a better innovativeness definition. For a good innovativeness metric, a reasonable question it should answer is how much novelty does the offspring of an individual add to the system. However the innovativeness metric used currently is expected novelty, which does not answer this question. It answers the question of how novel the children of these individual are. There can easily be a situation where the expected novelty of the offspring is high, but the novelty contributed to the system by the whole population of offspring is low. All of the children can be novel, but they might be novel the same way, so when considering the cumulative novelty added by them, it can be limited. One way we to go about creating a metric that would answer the question is by calculating the amount of entropy gained by adding all the offspring to the archive. The calculations would remain the same, the only difference would be to add some constant points (the novelty archive) when calculating the entropy. Because of the extra points, there is going to be some extra cost, which increases with the size of the archive.

5.6.3 Utilizing the full potential of evolvability

In most of the cases, we observed some improvements in the results compared to map elites without evolvability. However, these changes are incremental. In theory, evolvability should give the learning algorithm a very powerful tool. A tool to control exploration during learning by generalizing from past learning experience. However, in order for evolution of evolvability to be viable, three conditions need to be true (see chapter2 for a detailed explanation on the conditions of evolution of evolvability). If any one of these conditions is not true, we expect evolvability to not be useful.

One of the conditions is selection for evolvability. We fulfil this condition, since we added evolvability components to map elites which directly select for evolvability.

The second condition is that the data contains useful patterns, which predict which changes will be beneficial for future environments. Are there patterns in evolving neural networks for robotics locomotion which are useful for further evolution? Is the amount

of data enough from learning a single environment, or do we need to pool together data from multiple environments, so that they contain informative patterns? It is difficult to answer these questions without actually building an algorithm which utilizes evolvability. As of now, we do not have good answers to these questions.

The third condition is that our model needs a capacity to learn these evolvability patterns. This means the model needs to be able to control the kind of variation it generates. In our experiments, we used a feed forward neural network with directly encoded weights. The only way such a model can control its variation is by choosing to represent a policy in an area of the parameter space which has a good quality neighborhood. This means random variations in this neighborhood will result in adaptive changes. This is not a powerful control, thus this condition is very weakly fulfilled in our case. To improve the situation we need to use a model which is indirectly encoded, which we would like to do in the future. We discussed why an indirectly encoded model has more capacity to learn to be evolvable in chapter 2.

The situation is even more complicated as these are only necessary conditions, even if all of them are true, we still might fail. For example, there might exist evolvable points, but our learning algorithm cannot reach them, because the evolvability landscape is too difficult to learn for the evolvability algorithm.

Chapter 6

Indirect Encoding

6.1 Introduction

Most deep learning research is done with the natural representation of neural networks, where each weight in the network directly maps onto a separate parameter in the representation. We call this a direct encoding. On the other hand, in an indirect encoding, the weights do not directly map onto the representation, and instead, we apply a transformation to the representation to produce the weights. In Evolutionary Computation (EC), this transformation is commonly referred to as the genotype-phenotype mapping.

Direct encoding seems to work well and we can successfully train models with as many as 175 billion parameters [106]. As such, direct encoding dominates practically all benchmark problems. Natural evolution on the other hand uses indirect encoding. It is debatable whether evolution is as successful a problem solver because it uses indirect encoding or despite it. There could be many reasons why nature ended up with indirect encoding, be it biological limitations or because indirect encoding provides benefits. However, when designing our learning algorithms, we are faced with the decision of using either. This raises the question: Is there any advantage in using an indirect encoding?

6.1.1 The Difficulties with Indirect Encoding

As we discussed in chapter 2, indirect encodings have a powerful capability to control their own exploration during training. Since the ability to improve further has no effect on current fitness, greedy algorithms are not expected to select for representations which result in good exploration strategies. Because there are many more bad exploration strategies than good ones, if there is no selection for the ability to adapt, the exploration strategy will just drift, causing indirect encoding to most likely hurt learning performance. This leads to the main hypothesis of this chapter.

Hypothesis

Greedy learning algorithms are unlikely to make full use of the capabilities of indirect encoding.

We do not suggest however that it is impossible to make use of indirect encoding capabilities without selecting for the ability to adapt, only that it is much more difficult. Several researchers in the field of evolution of evolvability [15, 17, 31] argue that evolvability can emerge without selection in an unsupervised way. Huizinga et al. [23] showed that developmental canalization can emerge in a divergent search like environment.

Much effort was given to algorithms which instead of selecting for individuals with the ability to improve their fitness, select for the ability to generate diverse behaviour in their offspring [7, 26]. These algorithms capture a different aspect of evolvability which might be able to utilize the capabilities of indirect encoding just as well.

In the rest of the section, we evaluate our hypothesis in the context of past results with indirect encoding.

HyperNEAT

HyperNEAT [63] is one of the most well known indirect encoding techniques for neuroevolution. There are several demonstrations of how HyperNEAT outperforms direct encoding in different domains [107, 108, 109], especially if the task is more regular [110]. These re-

sults seemingly contradict our hypothesis, since HyperNEAT does not directly select for the ability to adapt. We argue however, that this is not the case for three reasons.

First, HyperNEAT uses innovation protection. Innovation protection was originally introduced in NEAT [1], and it keeps innovative genes in the gene pool even if they have a poor performance. The justification for it is that when we change the topology, fitness likely decreases first until the weights of the new structural elements can be fine-tuned. However, innovation protection also helps to protect individuals that are better at evolving, since it provides them with a few generations to prove their ability to improve. This means that HyperNEAT is actually indirectly selecting for the ability to adapt. More experiments are necessary to evaluate whether HyperNEAT would perform well without innovation protection, but this is beyond the scope of this thesis.

Second, the baselines for these results were using some version of NEAT such as plain NEAT, Fixed Topology NEAT (FT-NEAT) or Perceptron NEAT (P-NEAT). While NEAT might be a good algorithm to evolve the small query networks for HyperNEAT, it might not be an ideal baseline for the larger directly encoded networks. NEAT changes parameters one by one, recent results suggest that techniques which modify many weights at the same time perform much better for large networks, like CMA-ES [111], GA [77] and ES [32]. When we compare the performance of HyperNEAT to these more efficient baselines, on the task of learning to play Atari games, direct encoding seems to have the advantage in most games [32].

Finally, HyperNEAT experiments are typically using relatively small networks. Choromanska et al. [112] showed that as the network size increases, the number of bad quality local optima diminishes exponentially. The problem of local optima, which is a very important factor in the case of small networks, is less important for large networks. For this reason, the results obtained in many HyperNEAT experiments are not necessarily expected to generalize to large scale networks.

Differentiable Pattern Producing Network

One of the most impressive achievements of indirect encoding is the demonstrated ability to invent convolution from scratch using a fully connected architecture [61]. The DPPN

(Differentiable Pattern Producing Network) is a differentiable version of the CPPN (Compositional Pattern Producing Network) [113] used in HyperNEAT. In this work, the authors run experiments with three different settings. In the Darwinian setting, individuals were evaluated on their ability to solve the task without further adaptation. In the case of the Baldwinian setting, individuals were allowed to learn further by using gradient descent, resulting in selection for the ability to adapt. In the case of the Lamarckian setting, the situation is the same as with Baldwinian evolution with the additional feature of inheriting the learned weights as well. In the case of the Darwinian setting, without selection for the ability to adapt, the task was not solved successfully, not a single digit was recognisable in the image reconstruction task. When they used Baldwinian or Lamarckian evolution however performance was much better, and convolution-like fully connected weights were generated. This experiment supports our hypothesis that selection for the ability to learn is crucial in realizing the full capabilities of indirect encoding.

Hypernetworks

A recent indirect encoding technique called Hypernetworks [6] can achieve near state of the art performance on sequence modelling tasks. This result was achieved with dynamic Hypernetworks, where a recurrent network is enhanced with the new ability to modify its own weights during inference, based on the current input and state of the network. The simpler static Hypernetwork does not change the capabilities of the network, only the way the parameters are represented, making the comparison between direct and indirect encoding easy. When using static Hypernetworks to generate the weights of a convolutional network, the results on an image classification task are worse than direct encoding. This result however was obtained by using around an order of magnitude fewer parameters for the indirect encoding. We show in section 6.3.4. that we achieve similar results when using Hypernetworks with the same number of parameters, indirect encoding cannot outperform direct encoding on an image classification task with greedy learning.

6.2 Background

In this chapter we combine ideas from the fields of Deep Learning and Evolutionary Computation, so we use the terminology from both fields to describe similar concepts. For example, we use the terms evolvability and the ability to adapt which are similar concepts, both are concerned with potential yet unrealized improvements, but imply a different underlying algorithm. The situation is the same with the phrases "selecting for" or "optimizing for".

6.2.1 Indirect Encoding for Neuroevolution

There is a vast literature covering indirect network encoding. The field of artificial embryogeny is concerned with these techniques, a great review is available by [79]. Relatively few of these techniques were constructed with modern deep learning scale in mind (millions or billions of connections). In this section, we discuss two families of techniques, which were shown to be viable for these large networks.

One family of methods to indirectly encode the weights of a neural network is to use a query function and a substrate [63]. The query function is a parameterized function; typically a small neural network [113], which maps the coordinates of a source and a target neuron to a single weight between the source and target neuron. These coordinates for the neurons come from the substrate, which is often manually crafted by placing each neuron in 2D or 3D space, or it can also be learned [114]. A conceptual diagram of how this kind of encoding can represent weights can be seen in Fig. 6.1.

To calculate the weights of the whole network, we need to query this network as many times as there are weights . For a large network, this could mean millions of queries, which could become prohibitively expensive. This is especially problematic in cases when we only use the network a few times before updating, like supervised learning, and less problematic in control or reinforcement learning problems, where we use the network hundreds or thousands of times before updating it. This is only an issue during training since during inference time the network does not change anymore. Luckily the size of the query network is typically very small, and due to the large number of queries, they can

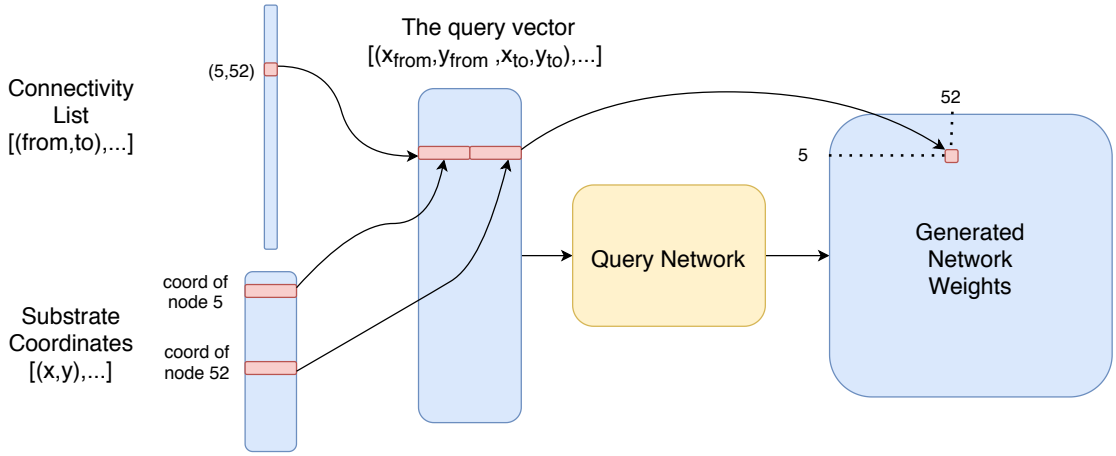


Figure 6.1: Query networks: In the case of query networks, each node in the neural network is assigned a coordinate in space, which is called the substrate (this example shows a 2D space with coordinates x and y). For each connection, a query vector is assembled from the coordinates of the source and target neurons. The weight for each connection is determined by evaluating the query vector with the query network, which is a small neural network. To generate the whole network as many forward passes are necessary as there are connections in the network. Yellow blocks represent the learned parameters, blue blocks represent fixed or generated values, and the red blocks show an example of how a single weight is generated

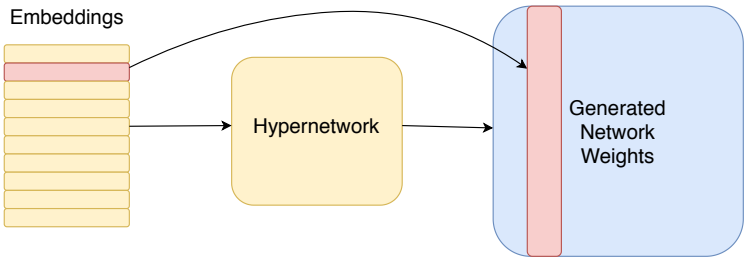


Figure 6.2: Hypernetworks: Each embedding is transformed into a chunk of weights by the Hypernetwork [6]. Yellow blocks represent the learned parameters, the blue block represents the generated weights, and the red blocks show an example of how a single embedding is transformed into a chunk of generated weights. See Fig. 6.4 on how a Hypernetwork can be used as part of a larger model

effectively utilize the GPU.

Another family of methods is to simply transform an embedding space to the weight space. The first method to use this technique for large neural networks is Hypernetworks [6] which uses a simple linear projection as transformation. A conceptual diagram of how this kind of encoding can represent weights can be seen in Fig. 6.2. The learned parameters are a set of embeddings and the parameters of the projection network. Each embedding is then used to generate a separate part of the network. This separation into smaller chunks is required to keep the size of the transformation manageable. Another side effect of this separation, which motivated the invention of the technique is that there is a kind of information sharing between these chunks since they are projected from the same subspace.

6.2.2 MAML

We described how MAML works in chapter 3. In this chapter, we used the gradient based version of MAML because both our models and the task are differentiable. The evolutionary version of the algorithm ES MAML [33] can be used in cases when either the model or the task or both are not differentiable. This property of ES MAML might be interesting for research into indirect encoding since there are many exotic and interesting nondifferentiable ways to represent networks [79].

6.3 Experiment

The goal of our experiments is to evaluate our original hypothesis, that indirect encoding is unlikely to be beneficial in the case of greedy learning, but can lead to better performance when the ability to adapt is selected.

We used two kinds of vision tasks, simple image classification on the FashionMNIST [115] dataset for greedy learning and few shot classification on the Omniglot [116] dataset for meta learning. The problem setting of few shot image classification is shown in Fig. 6.3

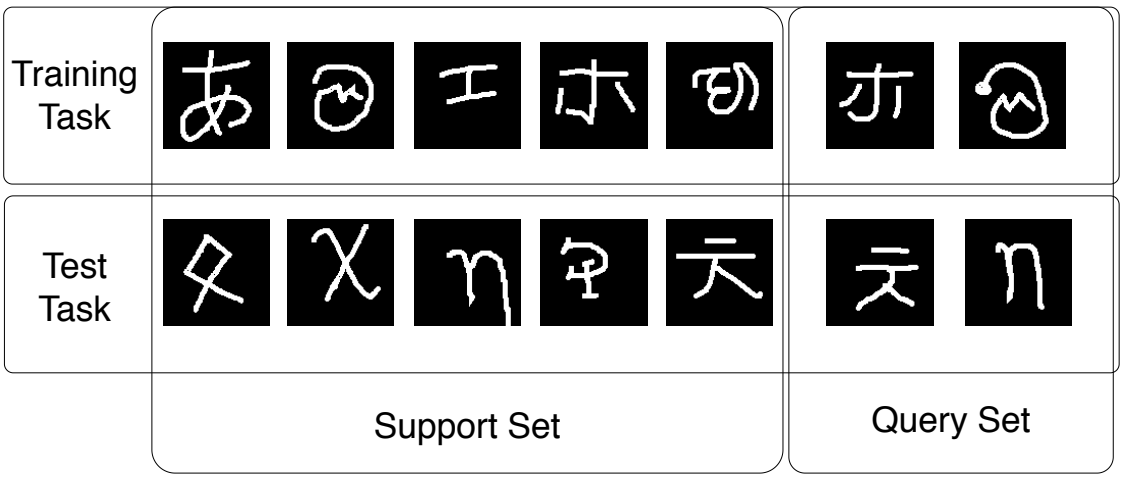


Figure 6.3: Few shot learning problem. The goal of MAML is to find model parameters that can be fine-tuned given the few examples in the support set (in this example 5 way 1 shot, there are 5 different classes with 1 example from each), so they can accurately classify the images in the query set. The training tasks were created by randomly sampling 5 out of the 1200 training classes. The performance is evaluated on the test tasks, which are created by sampling 5 out of the 400 test classes.

Table 6.1: Dimensions of the networks, and the number of parameters used in both direct and indirect encoding. The table also shows the hyperparameters used for indirect encoding.

	Hidden dims	Direct parameters	Indirect parameters	$[z_{dim1}, N_{in1}, N_{out1}]$	$[z_{dim2}, N_{in2}, N_{out2}]$
Tiny	[32,16]	25,829	25,977	[14,2,2]	[16,2,2]
Small	[64,32]	52,677	51,237	[14,4,4]	[16,2,2]
Medium	[128,64]	109,445	107,229	[30,4,4]	[16,2,2]
Large	[256,128,64,64]	247,621	244,837	[56,4,4]	[32,4,4]

We used fully connected networks because convolutional networks are already very good at vision tasks and we wanted to leave room for improvement. Because the two dataset uses the same resolution 28 by 28 we could use the same networks without any modification for both tasks.

6.3.1 Fair Comparison

To determine whether indirect encoding is beneficial for learning, we need to establish a baseline with direct encoding. To make the comparison fair, we used approximately the same number of parameters to encode the exact same networks with both direct and indirect encoding. We used 4 different sized networks, which are summed up in Table 6.1.

For indirect encoding, we generated the weights of the first two hidden layers (the

vast majority of parameters), the biases and the rest of the weights were encoded directly, as shown in Fig. 6.4. The authors of the original Hypernetwork paper used a single Hypernetwork to generate the weights of all layers. They argue that this constrained the system to share some commonality between the layers, which resulted in decreased performance [6]. This would especially be the case for fully connected networks since the intermediate fully connected representations lack the common structure that the intermediate representations of convolutional networks have. For this reason, we used separate Hypernetworks for the two generated layers.

6.3.2 Implementation Details

We used the same formulation of the Hypernetwork as in [6], but instead of a simple projection, 2 matrix multiplications are used to project the embeddings into weights. First, the embeddings with size z_{dim} are projected into the shape N_{in} by z_{dim}. The second projection then projects the result of the previous projection into the shape [N_{out},N_{in},$unit_{dim}$], where $unit_{dim}$ is the size of the smallest chunk of weights generated. N_{out},N_{in} are hyperparameters controlling how many weights should be generated from a simple embedding. Doing the projection this way is equivalent to a simple large projection but uses way fewer parameters because the weights of the second matrix are reused many times. We choose $unit_{dim}$ to be the number of connections a single neuron has in the generated layer.

Initializing weights is an important aspect of training neural networks to avoid the vanishing or exploding gradient problem. Normally we would want to initialize our weight in a way that the magnitude of the activations throughout the network stays constant. This can be achieved by initializing each layer so their gain is one [117]. Normally in a fully connected layer, the variance of the activations in a layer depends on the variance of the inputs, the variance of the weights, and the number of neurons in the previous layer as shown in equation 6.1.

$$Var(a_l) = n_{l-1} * Var(W_l) * Var(a_{l-1}) \tag{6.1}$$

In the case of Hypernetworks however, we use one network to generate the weights of another network. We initialized the Hypernetwork in a way that will result in the generated

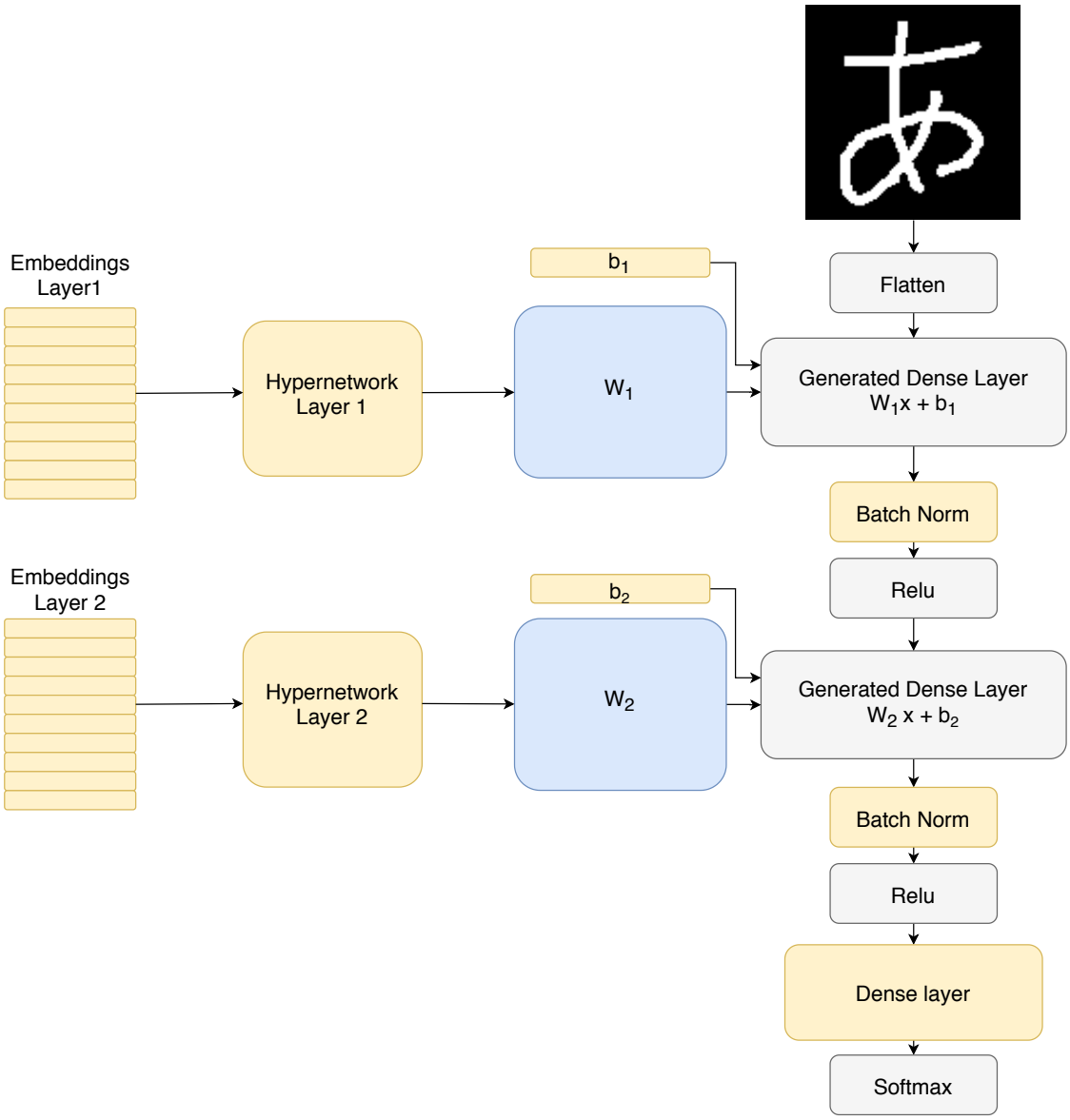


Figure 6.4: The indirect architecture used in the experiments. The yellow boxes are learned parameters, the blue boxes are generated parameters and the grey boxes are functions. The direct encoding uses the same architecture, but the two dense layers are represented directly.

layer having a gain of one. In the case of Hypernetworks, because the matrix weights in the second projection are reused multiple times, we need to use the number of neurons contributing to a single weight, which is the number of embeddings, z_{dim} (equation 6.3). In the following equations, W_1 is the first matrix and W_2 is the second matrix in the Hypernetwork.

$$Var(a_1) = z_{dim} * Var(W_1) * Var(a_0) \tag{6.2}$$

$$Var(a_2) = z_{dim} * Var(W_2) * Var(a_1) \tag{6.3}$$

$$Var(a_2) = z_{dim} * Var(W_2) * z_{dim} * Var(W_1) * Var(a_0) \tag{6.4}$$

Let the gain of the Hypernetwork be equal to the required variance of the generated layer for it to have a gain of one, and provide the additional constraint of $Var(W_1) = Var(W_2)$

$$Var(a_2)/Var(a_0) := 1/n_{generated\ fan\ in} \tag{6.5}$$

$$Var(W) = \sqrt{\frac{1}{n_{generated\ fan\ in} * z_{dim} * z_{dim}}} \tag{6.6}$$

The source code for all of our experiments is available at `https://github.com/adam-katona/indirect_encoding_maml`

6.3.3 Greedy Learning Experiment

To evaluate the effect of indirect encoding on performance in the case of a greedy learning algorithm, we used the FashionMNIST dataset for image classification. The networks were trained with gradient descent. The batch size was 64, we used the Adam optimizer with a learning rate of 0.001. Each run was repeated 20 times, test results are shown in Fig. 6.5 and in Table 6.2.

Indirect encoding achives smaller median test accuracy on the Small (Median difference: 0.23%, Mann-Whitney U Test; p=0.0021), Medium (Median difference: 0.41%, Mann-Whitney U Test; p=0.0002), and Large (Median difference: 0.33%, Mann-Whitney U Test; p=1.77e-06) network sizes. For the tiny (Median difference: 0.04%, Mann-Whitney U Test; p=0.1848) network, the difference is not statistically significant.

These are similar results reported in [6], showing the inability of greedy learning to

Table 6.2: Median test accuracies (out of 20 runs) achieved on the FashionMNIST dataset.

	Direct	Indirect
Tiny	**0.8723**	0.8718
Small	**0.8840**	0.8817
Medium	**0.8903**	0.8862
Large	**0.8937**	0.8904

benefit from the capabilities of indirect encoding, supporting our original hypothesis.

6.3.4 Meta Learning Experiment

To evaluate the performance of indirect encoding when we are optimizing for the ability to adapt, we run experiments with few shot learning on the Omniglot dataset. We use 5-way 1 shot learning. We followed the procedure described in [25]. We used 1200 characters for training and 400 for testing. We augmented the characters by applying multiples of 90° rotations. We used batch normalization in the same way as in the original MAML implementation, only using batch statistics and not accumulating running statistics. We used a learnable per step, per parameter learning rate for the fine-tuning update, as proposed in [66]. We used cosine annealing meta learning rate schedule, as proposed in [66]. We used a meta batch size of 32. We used the Adam optimizer and the initial meta learning rate of 0.005 using cosine annealing learning rate scheduler with a restart period of 3000 meta batches. We trained each model for 50000 iterations (meta batches). We used a single adaptation step while training, and used three while evaluating performance on the test tasks, the same way as done in the original MAML paper [25].

Each run was repeated 8 times, test results are shown in Fig. 6.6 and in Table 6.3. For the Small (Median difference: 3.11%, Mann-Whitney U Test; p=0.0002) and Medium (Median difference: 2.16%, Mann-Whitney U Test; p=0.0104) network sizes, indirect encoding achieves higher accuracies. For the Tiny (Median difference: 1.55%, Mann-Whitney U Test; p=0.1049) and Large (Median difference: -0.31%, Mann-Whitney U Test; p=0.3357) network sizes, the difference is not statistically significant.

The result with the large network is surprising since for all other network configurations indirect encoding had the advantage. For the large network, we added 2 additional directly encoded layers. We suspect that there is some kind of interesting interactions

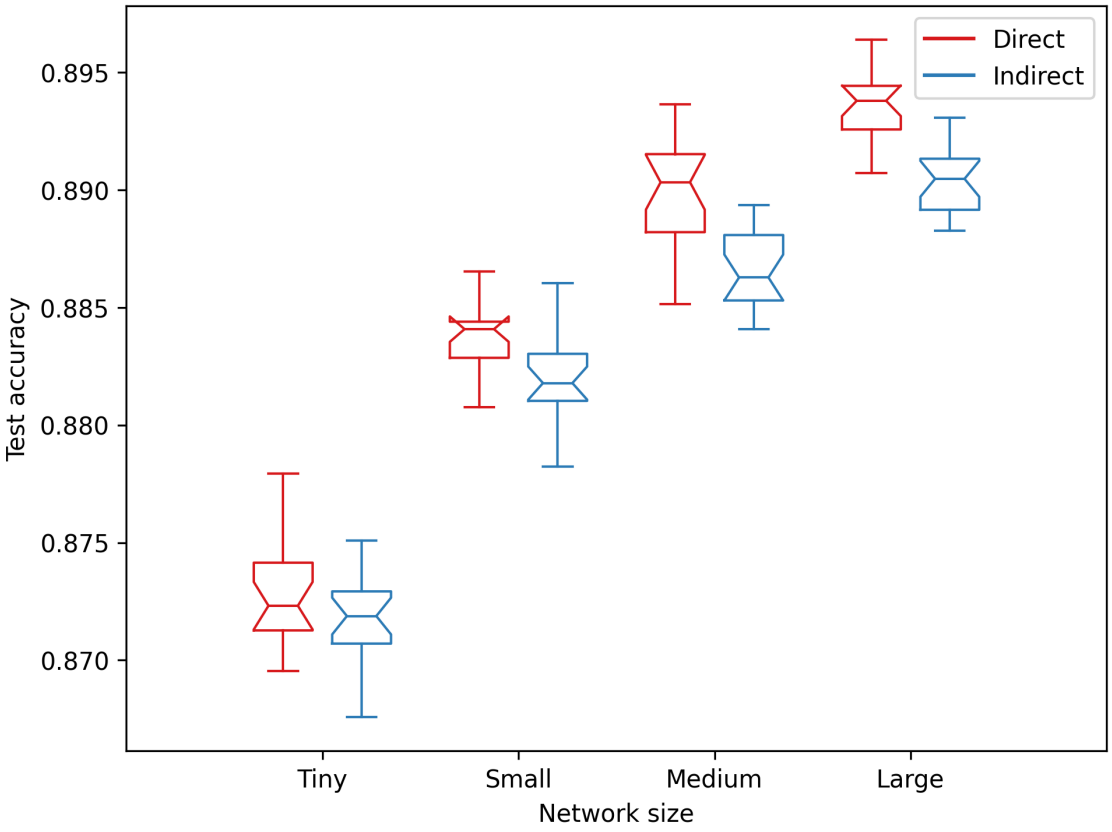


Figure 6.5: Greedy learning results: Final test accuracy on the FashionMNIST dataset. Without selecting for adaptability, direct encoding slightly but consistently outperforms indirect encoding in a fair comparison. The difference is statistically significant for the small, medium and large networks ($p<0.01$, Mann-Whitney U test)

Table 6.3: Median test accuracies (out of 8 runs) achieved on 5-way, 1-shot learning on the Omniglot dataset.

	Direct	Indirect
Tiny	0.739	**0.754**
Small	0.775	**0.806**
Medium	0.818	**0.839**
Large	**0.875**	0.872

between the direct and indirect layers which hinders performance.

6.4 Conclusion

We proposed the hypothesis that greedy learning is unlikely to benefit from the capabilities of indirect encoding, and selecting for the ability to adapt is necessary. We verified the previously demonstrated [6] results that the indirect encoding technique Hypernetworks achieves lower accuracy when trained on an image classification task compared to direct encoding. We then showed that when the ability to adapt is selected with MAML, Hypernetworks can outperform direct encoding on an image classification task. Our results suggest that optimizing for the ability to adapt is indeed of key importance when learning with indirect encoding. More experiments are needed in different domains to verify whether the hypothesis holds in a more general setting. We hope that our results will motivate other researchers to explore the exciting possibilities of utilizing meta learning to realize the powerful potential of indirect encoding.

Subsequent to the release of the study upon which this chapter is founded, an additional research paper co-authored by the author of this thesis has provided further evidence for the hypothesis of this chapter. Barabasi et al. [118] presented findings demonstrating that a different biologically inspired indirect encoding also outperforms direct encoding with meta learning in few shot image classification, while not performing better in the case of greedy learning.

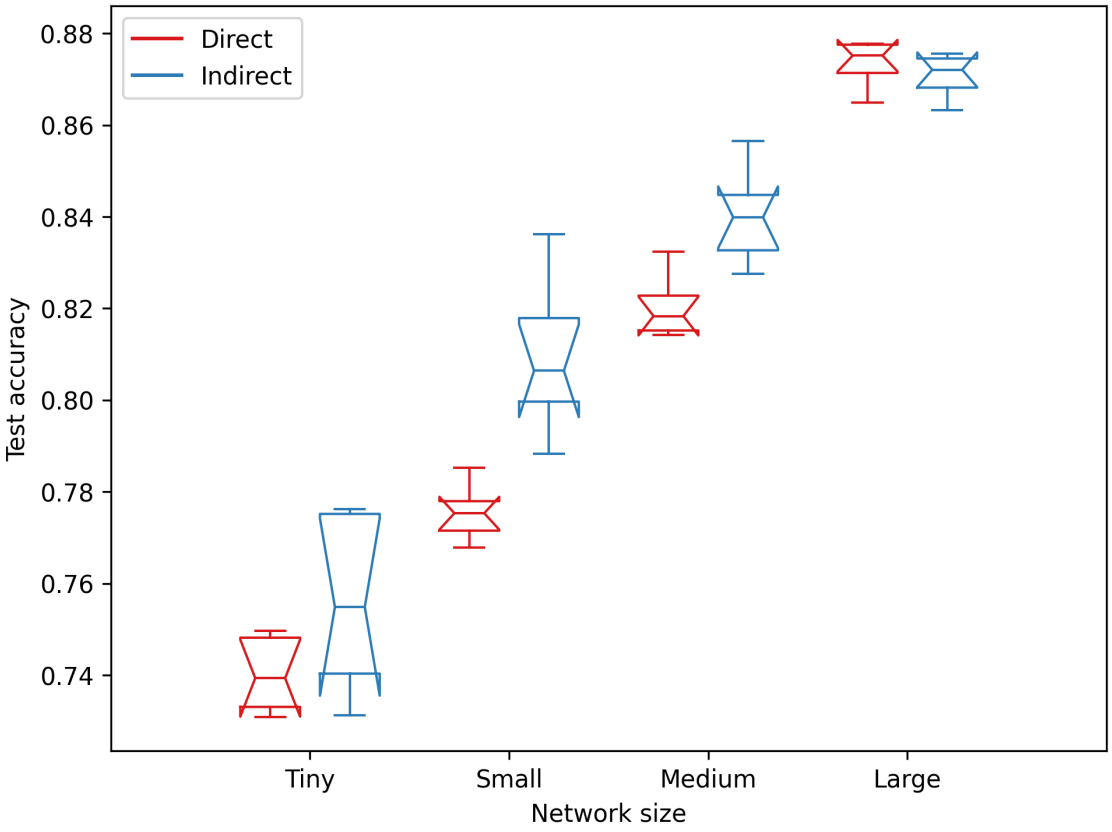


Figure 6.6: Meta learning results: Final test accuracies after the third gradient step on the test tasks of 5-shot, 1-way image classification. When adaptability is selected, the indirect encoding outperforms the direct encoding in a fair comparison for small and medium network sizes.

Chapter 7

Conclusion

This thesis investigated the necessary conditions for evolution of evolvability in the context of neuroevolution. This chapter presents a summary of the main contributions and their implications for the wider research field.

7.1 Contributions

7.1.1 Evolvability algorithm with linear sample complexity

We demonstrated the feasibility of developing an algorithm that optimizes for expected evolvability with a linear $O(n)$ sample complexity, where n represents the population size. We introduced Resampling Expected Evolvability ES (REE-ES) as a solution to achieve this goal. The algorithm can achieve $O(n)$ scaling by reusing common samples to estimate the evolvability of the whole population. The insight for this algorithm is the realization that there is a large overlap between the distributions of the children of each individual in the population in the case of the ES algorithm. However, while the approach is intriguing, it scales poorly with respect to the problem dimension. As a result, the practical applications of the algorithm may be limited.

7.1.2 Quality evolvability: focusing evolvability for solving problems

We introduced a new approach to evolvability optimization called Quality Evolvability, and the algorithm Quality Evolvability ES. The QE approach aims to find individuals with a diverse and well-performing distribution of offspring. Unlike previous evolvability algorithms that only focused on evolvability and ignored fitness, the QE approach considers both evolvability and fitness. While looking for evolvability alone may lead to interesting solutions to some tasks, this is not the case for all tasks. With Quality Evolvability, we may utilize evolvability for problems where evolvability and fitness are not aligned. The reverse is also true, looking for fitness can help the algorithm achieve higher evolvability.

7.1.3 Evolvability Map Elites: Exploiting synergies between quality, evolvability and diversity

The synergistic relationship between quality and diversity is well known, the established area of quality diversity algorithms demonstrated that looking for quality and diversity together can result in higher quality and diversity compared to looking for them separately. In chapter 4, we explored the synergistic relationship between quality and evolvability.

These approaches can result in higher objective scores despite dividing their attention between multiple objectives because looking for one objective might lead to stepping stones which can be used to bootstrap the search for the other objective. The intuition why this is the case is that most hard problems are deceptive in nature [36], the steps to solve these problems are not obvious beforehand thus a simple objective function is not suitable for measuring progress towards that objectives.

In chapter 5 we attempted to further exploit this phenomenon by introducing Evolvability Map Elites, which aims the exploit the three-way synergies between quality diversity and evolvability. We concluded that while the approach resulted in limited success, sometimes resulting in higher scores in some of the objectives, the improvements were not transformational. The results suggest that dividing the attention of the algorithm in too many directions often lessens the ability of the algorithm to exploit these kinds of synergies.

Moving forward, we plan to enhance our approach by introducing an adaptive version of the algorithm that can determine which objectives to focus on to maximize learning. This will prevent the algorithm from being spread too thin across too many objectives.

7.1.4 Indirect encoding is unlikely to work with greedy learning algorithms

We discussed that indirect encoding is a critical component of evolution of evolvability. Indirect encoding allows evolution to make changes in some directions easy and other directions difficult, therefore biasing learning towards more promising areas of the search space. In chapter 6, we proposed the hypothesis that indirect encoding is unlikely to provide any benefit when used with greedy learning algorithms. We provided some evidence for this hypothesis in the context of an image classification task.

While both indirect encoding and evolvability/meta-learning are areas of active research, there has been limited exploration of applying higher-level learning algorithms to indirect encoding. In particular, there are few studies that use algorithms which directly select for evolvability/adaptability, compared to relying on some kind of indirect selection effect.

We argue that the success of indirect encoding can be significantly enhanced through greater attention to the application of these types of algorithms to indirect encoding techniques.

7.2 Limitations / Future work

7.2.1 More powerful indirect encodings

In this thesis, we either used direct encoding or a simple linear transformation as the encoding. In order for a system to learn evolvability, it needs a powerful indirect encoding which can learn the biases that make future learning more effective. The encoding must provide sufficient capacity for learning these biases. There are multiple approaches for creating such indirect encodings.

One approach is to use some kind of complex mathematical transformation as the encoding. Two examples of such methods are using a neural network to generate parameters [6], or using a neural network which is queried to generate parameters [63].

Another approach is instead of trying to invent some complex function, we simulate some simple rules, and let the interaction of these rules creates some complex structure. This is the kind of encoding nature uses. An example of such an encoding is the Cellular Automata [119] or the more expressive Neural Cellular Automata [120].

7.2.2 The importance of the choice of the behaviour characterization

The evolvability algorithms presented in this thesis all depend on the behaviour characterization function to quantify evolvability. These algorithms are very sensitive to the choice of this function. It is not obvious how to choose a behaviour characterization function that will work for a certain task. This is a problem for two reasons.

1. It makes it difficult to apply these algorithms to new problems, as the experimenter needs to choose a behaviour characterization function.

2. Even if a behavior characterization function produces results, it is difficult to determine if the choice of this function was good, or if much better alternatives exist.

A system which learns to quantify evolvability on its own would not require the user to come up with clever behaviour characterization and would be easier to apply to a wide range of problems. A possible way to cunstruct such a system would involve two steps.

The first step is to automatically come up behavior characteization functions. One way to achieve this is to use dimensionality reduction techniques, to learn behavior characterizations from data in an unsupervised way. These methods allow us to find a low dimesional representation of the data in a way which preserves the most variance or resut is the most efficient compresses. For example Cully et al. [121] used an autoencoder to derive a latent representation for a robot arm trajectory.

The second step would be to automatically select the best behavior characterization function, by evaluating how well they work for our evolvability algorithms. The obvious

problem with doing this naively is it would increase the required computational resources several orders of magnitude.

Bibliography

[1] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[2] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Frontiers in Robotics and AI*, p. 40, 2016.

[3] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, 2015.

[4] J. Lehman and K. O. Stanley, "Evolving a diversity of virtual creatures through novelty search and local competition," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pp. 211–218, 2011.

[5] A. Katona, D. W. Franks, and J. A. Walker, "Quality Evolvability ES: Evolving Individuals With a Distribution of Well Performing and Diverse Offspring," in *ALIFE 2021: The 2021 Conference on Artificial Life*, MIT Press, 2021.

[6] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," *arXiv preprint arXiv:1609.09106*, 2016.

[7] A. Gajewski, J. Clune, K. O. Stanley, and J. Lehman, "Evolvability ES: scalable and direct optimization of evolvability," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 107–115, 2019.

[8] A. Katona, N. Lourenço, P. Machado, D. W. Franks, and J. A. Walker, "Utilizing the untapped potential of indirect encoding for neural networks with meta learning," in *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pp. 537–551, Springer, 2021.

[9] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[10] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[11] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[12] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022.

[13] A. Ziegler, E. Kalliamvakou, X. A. Li, A. Rice, D. Rifkin, S. Simister, G. Sittampalam, and E. Aftandilian, "Productivity assessment of neural code completion," in *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, pp. 21–29, 2022.

[14] Y. Bengio, Y. Lecun, and G. Hinton, "Deep learning for AI," *Communications of the ACM*, vol. 64, no. 7, pp. 58–65, 2021.

[15] M. Pigliucci, "Is evolvability evolvable?," *Nature Reviews Genetics*, vol. 9, no. 1, pp. 75–82, 2008.

[16] J. L. Payne and A. Wagner, "The causes of evolvability and their evolution," *Nature Reviews Genetics*, vol. 20, no. 1, pp. 24–38, 2019.

[17] R. A. Watson and E. Szathmáry, "How can evolution learn?," *Trends in ecology & evolution*, vol. 31, no. 2, pp. 147–157, 2016.

[18] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artificial intelligence review*, vol. 18, pp. 77–95, 2002.

[19] T. Taylor, M. Bedau, A. Channon, D. Ackley, W. Banzhaf, G. Beslon, E. Dolson, T. Froese, S. Hickinbotham, T. Ikegami, *et al.*, "Open-ended evolution: Perspectives from the OEE workshop in York," *Artificial life*, vol. 22, no. 3, pp. 408–423, 2016.

[20] R. Wang, J. Lehman, J. Clune, and K. O. Stanley, "Paired open-ended trailblazer (POET): Endlessly generating increasingly complex and diverse learning environments and their solutions," *arXiv preprint arXiv:1901.01753*, 2019.

[21] J. Zhang, J. Lehman, K. Stanley, and J. Clune, "Omni: Open-endedness via models of human notions of interestingness," *arXiv preprint arXiv:2306.01711*, 2023.

[22] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[23] J. Huizinga, K. O. Stanley, and J. Clune, "The emergence of canalization and evolvability in an open-ended, interactive evolutionary system," *Artificial life*, vol. 24, no. 3, pp. 157–181, 2018.

[24] D. Tarapore and J.-B. Mouret, "Evolvability signatures of generative encodings: beyond standard performance benchmarks," *Information Sciences*, vol. 313, pp. 43–61, 2015.

[25] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*, pp. 1126–1135, PMLR, 2017.

[26] H. Mengistu, J. Lehman, and J. Clune, "Evolvability search: directly selecting for evolvability in order to study and produce it," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pp. 141–148, 2016.

[27] P. W. Anderson, "More is different: broken symmetry and the nature of the hierarchical structure of science.," *Science*, vol. 177, no. 4047, pp. 393–396, 1972.

[28] T. Flatt, "The evolutionary genetics of canalization," *The Quarterly review of biology*, vol. 80, no. 3, pp. 287–316, 2005.

[29] S. McCouch, "Diversifying selection in plant breeding," *PLoS biology*, vol. 2, no. 10, p. e347, 2004.

[30] G. P. Wagner and L. Altenberg, "Perspective: complex adaptations and the evolution of evolvability," *Evolution*, vol. 50, no. 3, pp. 967–976, 1996.

[31] L. Altenberg *et al.*, "The evolution of evolvability in genetic programming," *Advances in genetic programming*, vol. 3, pp. 47–74, 1994.

[32] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.

[33] X. Song, W. Gao, Y. Yang, K. Choromanski, A. Pacchiano, and Y. Tang, "ES-MAML: Simple hessian-free meta learning," *arXiv preprint arXiv:1910.01215*, 2019.

[34] E. Conti, V. Madhavan, F. Petroski Such, J. Lehman, K. Stanley, and J. Clune, "Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents," *Advances in neural information processing systems*, vol. 31, 2018.

[35] J. Lehman, J. Clune, D. Misevic, C. Adami, L. Altenberg, J. Beaulieu, P. J. Bentley, S. Bernard, G. Beslon, D. M. Bryson, *et al.*, "The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities," *Artificial life*, vol. 26, no. 2, pp. 274–306, 2020.

[36] J. Lehman and K. O. Stanley, "Abandoning objectives: Evolution through the search for novelty alone," *Evolutionary computation*, vol. 19, no. 2, pp. 189–223, 2011.

[37] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, "Large-scale study of curiosity-driven learning," *arXiv preprint arXiv:1808.04355*, 2018.

[38] J. Brookfield, "Evolution: the evolvability enigma," *Current Biology*, vol. 11, no. 3, pp. R106–R108, 2001.

[39] M. Conrad, "The geometry of evolution," *BioSystems*, vol. 24, no. 1, pp. 61–81, 1990.

[40] J. D. Bloom, S. T. Labthavikul, C. R. Otey, and F. H. Arnold, "Protein stability promotes evolvability," *Proceedings of the National Academy of Sciences*, vol. 103, no. 15, pp. 5869–5874, 2006.

[41] J. Clune, J.-B. Mouret, and H. Lipson, "The evolutionary origins of modularity," *Proceedings of the Royal Society B: Biological sciences*, vol. 280, no. 1755, p. 20122863, 2013.

[42] R. Riedl, "A systems-analytical approach to macro-evolutionary phenomena," *The Quarterly Review of Biology*, vol. 52, no. 4, pp. 351–370, 1977.

[43] S. Okasha, "Does the concept of "clade selection" make sense?," *Philosophy of Science*, vol. 70, no. 4, pp. 739–751, 2003.

[44] W. D. Hamilton, "The genetical evolution of social behaviour. ii," *Journal of theoretical biology*, vol. 7, no. 1, pp. 17–52, 1964.

[45] J. Clune, D. Misevic, C. Ofria, R. E. Lenski, S. F. Elena, and R. Sanjuán, "Natural selection fails to optimize mutation rates for long-term adaptation on rugged fitness landscapes," *PLoS Computational Biology*, vol. 4, no. 9, p. e1000187, 2008.

[46] C. O. Wilke, J. L. Wang, C. Ofria, R. E. Lenski, and C. Adami, "Evolution of digital organisms at high mutation rates leads to survival of the flattest," *Nature*, vol. 412, no. 6844, pp. 331–333, 2001.

[47] M. Lynch, "The frailty of adaptive hypotheses for the origins of organismal complexity," *Proceedings of the National Academy of Sciences*, vol. 104, no. suppl 1, pp. 8597–8604, 2007.

[48] D. Simon, *Evolutionary optimization algorithms*. John Wiley & Sons, 2013.

[49] W. Zhong and N. K. Priest, "Stress-induced recombination and the mechanism of evolvability," *Behavioral ecology and sociobiology*, vol. 65, no. 3, pp. 493–502, 2011.

[50] R. Dawkins, "The evolution of evolvability," *On growth, form and computers*, pp. 239–255, 2003.

[51] E. E. Goldberg, J. R. Kohn, R. Lande, K. A. Robertson, S. A. Smith, and B. Igić, "Species selection maintains self-incompatibility," *Science*, vol. 330, no. 6003, pp. 493–495, 2010.

[52] J. Lehman and R. Miikkulainen, "Enhancing divergent search through extinction events," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 951–958, ACM, 2015.

[53] A. M. Nguyen, J. Yosinski, and J. Clune, "Innovation engines: Automated creativity

and improved stochastic optimization via deep learning," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 959–966, 2015.

[54] H. Janmohamed, T. Pierrot, and A. Cully, "Improving the data efficiency of multi-objective quality-diversity through gradient assistance and crowding exploration," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 165–173, 2023.

[55] S. Risi and K. O. Stanley, "Improving deep neuroevolution via deep innovation protection," *arXiv preprint arXiv:2001.01683*, 2019.

[56] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, "Natural evolution strategies," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 949–980, 2014.

[57] E. Conti, V. Madhavan, F. P. Such, J. Lehman, K. O. Stanley, and J. Clune, "Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents," *arXiv preprint arXiv:1712.06560*, 2017.

[58] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in *Proceedings of IEEE international conference on evolutionary computation*, pp. 312–317, IEEE, 1996.

[59] J. Huizinga, J. Clune, and J.-B. Mouret, "Evolving neural networks that are both modular and regular: Hyperneat plus the connection cost technique," in *Proceedings of the 2014 annual conference on genetic and evolutionary computation*, pp. 697–704, 2014.

[60] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, 2019.

[61] C. Fernando, D. Banarse, M. Reynolds, F. Besse, D. Pfau, M. Jaderberg, M. Lanctot, and D. Wierstra, "Convolution by evolution: Differentiable pattern producing networks," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pp. 109–116, 2016.

[62] R. Sutton, "The bitter lesson," *Incomplete Ideas (blog) www.incompleteideas.net/IncIdeas/BitterLesson.html*, vol. 13, no. 1, 2019.

[63] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artificial life*, vol. 15, no. 2, pp. 185–212, 2009.

[64] A. Nichol and J. Schulman, "Reptile: a scalable metalearning algorithm," *arXiv preprint arXiv:1803.02999*, vol. 2, no. 3, p. 4, 2018.

[65] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-sgd: Learning to learn quickly for few-shot learning," *arXiv preprint arXiv:1707.09835*, 2017.

[66] A. Antoniou, H. Edwards, and A. Storkey, "How to train your maml," *arXiv preprint arXiv:1810.09502*, 2018.

[67] R. Houthooft, Y. Chen, P. Isola, B. Stadie, F. Wolski, O. Jonathan Ho, and P. Abbeel, "Evolved policy gradients," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[68] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[69] J. Chen, X. Qiu, P. Liu, and X. Huang, "Meta multi-task learning for sequence modeling," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[70] J. Lehman, J. Chen, J. Clune, and K. O. Stanley, "Es is more than just a traditional finite-difference approximator," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 450–457, 2018.

[71] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[72] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[73] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[74] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[75] J. Lehman, J. Chen, J. Clune, and K. O. Stanley, "Safe mutations for deep and recurrent neural networks through output gradients," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 117–124, 2018.

[76] N. J. Radcliffe, "Genetic set recombination and its application to neural network topology optimisation," *Neural Computing & Applications*, vol. 1, pp. 67–90, 1993.

[77] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *arXiv preprint arXiv:1712.06567*, 2017.

[78] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[79] K. O. Stanley and R. Miikkulainen, "A taxonomy for artificial embryogeny," *Artificial life*, vol. 9, no. 2, pp. 93–130, 2003.

[80] S. Risi, J. Lehman, and K. O. Stanley, "Evolving the placement and density of neurons in the hyperneat substrate," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 563–570, 2010.

[81] S. Cussat-Blanc, K. Harrington, and J. Pollack, "Gene regulatory network evolution through augmenting topologies," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 6, pp. 823–837, 2015.

[82] Google, "Xla: Accelerated linear algebra." `https://www.tensorflow.org/xla`, 2021.

[83] R. Frostig, M. J. Johnson, and C. Leary, "Compiling machine learning programs via high-level tracing," *Systems for Machine Learning*, vol. 4, no. 9, 2018.

[84] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, *et al.*, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[85] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, "Brax–a differentiable physics engine for large scale rigid body simulation," *arXiv preprint arXiv:2106.13281*, 2021.

[86] J.-B. Mouret, "Novelty-based multiobjectivization," in *New horizons in evolutionary robotics*, pp. 139–154, Springer, 2011.

[87] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[88] J. Lehman and K. O. Stanley, "Novelty search and the problem with objectives," *Genetic programming theory and practice IX*, pp. 37–56, 2011.

[89] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[90] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Conference on robot learning*, pp. 1094–1100, PMLR, 2020.

[91] C. Colas, V. Madhavan, J. Huizinga, and J. Clune, "Scaling map-elites to deep neuroevolution," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 67–75, 2020.

[92] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv preprint arXiv:1504.04909*, 2015.

[93] A. Alvarez, S. Dahlskog, J. Font, and J. Togelius, "Empowering quality diversity in dungeon design with interactive constrained map-elites," in *2019 IEEE Conference on Games (CoG)*, pp. 1–8, IEEE, 2019.

[94] M. Balla, A. Barahona-Rıos, A. Katona, N. P. Pérez, and R. Spick, "Illuminating game space using map-elites for assisting video game design," in *11th AISB Symposium on AI & Games (AI&G)*, pp. 1–6, 2021.

[95] J. J. Randolph and R. Bednarik, "Publication bias in the computer science education research literature.," *J. Univers. Comput. Sci.*, vol. 14, no. 4, pp. 575–589, 2008.

[96] V. Vassiliades, K. Chatzilygeroudis, and J.-B. Mouret, "Scaling up map-elites using Centroidal Voronoi Tessellations," *arXiv preprint arXiv:1610.05729*, 2016.

[97] O. Nilsson and A. Cully, "Policy gradient assisted map-elites," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 866–875, 2021.

[98] N. Rakicevic, A. Cully, and P. Kormushev, "Policy manifold search: Exploring the manifold hypothesis for diversity-based neuroevolution," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 901–909, 2021.

[99] M. C. Fontaine, J. Togelius, S. Nikolaidis, and A. K. Hoover, "Covariance matrix adaptation for the rapid illumination of behavior space," in *Proceedings of the 2020 genetic and evolutionary computation conference*, pp. 94–102, 2020.

[100] A. Cully, "Multi-emitter map-elites: improving quality, diversity and data efficiency with heterogeneous sets of emitters," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 84–92, 2021.

[101] T. Pierrot, G. Richard, K. Beguir, and A. Cully, "Multi-objective quality diversity optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 139–147, 2022.

[102] L. Cazenille, "Ensemble feature extraction for multi-container quality-diversity algorithms," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 75–83, 2021.

[103] A. Cully and Y. Demiris, "Quality and diversity optimization: A unifying modular framework," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 245–259, 2017.

[104] A. Coninx and S. Doncieux, "Younger is better: A simple and efficient selection strategy for map-elites," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 87–88, 2021.

[105] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A llvm-based python jit compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pp. 1–6, 2015.

[106] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[107] J. Clune, C. Ofria, and R. T. Pennock, "The sensitivity of hyperneat to different geometric representations of a problem," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pp. 675–682, 2009.

[108] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro, "Using gp is neat: Evolving compositional pattern production functions," in *European Conference on Genetic Programming*, pp. 3–18, Springer, 2018.

[109] J. Gauci and K. O. Stanley, "A case study on the critical role of geometric regularity in machine learning.," in *AAAI*, pp. 628–633, 2008.

[110] J. Clune, C. Ofria, and R. T. Pennock, "How a generative encoding fares as problem-regularity decreases," in *International Conference on Parallel Problem Solving from Nature*, pp. 358–367, Springer, 2008.

[111] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, "A neuroevolution approach to general atari game playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 355–366, 2014.

[112] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The loss surfaces of multilayer networks," in *Artificial intelligence and statistics*, pp. 192–204, 2015.

[113] K. O. Stanley, "Compositional pattern producing networks: A novel abstraction of development," *Genetic programming and evolvable machines*, vol. 8, no. 2, pp. 131–162, 2007.

[114] S. Risi and K. O. Stanley, "Enhancing ES-Hyperneat to evolve more complex regular neural networks," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pp. 1539–1546, 2011.

[115] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[116] B. Lake, R. Salakhutdinov, J. Gross, and J. Tenenbaum, "One shot learning of simple visual concepts," in *Proceedings of the annual meeting of the cognitive science society*, vol. 33, 2011.

[117] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[118] D. L. Barabási, T. Beynon, Á. Katona, and N. Perez-Nieves, "Complex computation from developmental priors," *Nature Communications*, vol. 14, no. 1, p. 2226, 2023.

[119] J. Von Neumann, A. W. Burks, *et al.*, "Theory of self-reproducing automata," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 3–14, 1966.

[120] A. Mordvintsev, E. Randazzo, E. Niklasson, and M. Levin, "Growing neural cellular automata," *Distill*, vol. 5, no. 2, p. e23, 2020.

[121] A. Cully and Y. Demiris, "Hierarchical behavioral repertoires with unsupervised descriptors," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 69–76, 2018.