Sorbonne Université
InstaDeep - BioNTech

Thèse de Doctorat

Spécialité **Informatique**

présentée par
Valentin Macé

Exploration Fondée sur la Diversité pour l'Apprentissage par Renforcement Profond

---

sous la direction d' **Olivier Sigaud** et **Nicolas Perrin-Gilbert**.

Soutenue publiquement le **18 octobre 2024** à **Paris** devant le jury composé de

M. Jean-Baptiste **Mouret**	Inria	Rapporteur
M. Emmanuel **Rachelson**	ISAE-SUPAERO	Rapporteur
M. Cédric **Colas**	Inria & MIT	Examinateur
M. Stéphane **Doncieux**	Sorbonne Université & CNRS	Examinateur
M^{me} Sao Mai **Nguyen**	ENSTA Paris	Examinatrice
M. Olivier **Sigaud**	Sorbonne Université	Directeur de thèse
M. Nicolas **Perrin-Gilbert**	Sorbonne Université & CNRS	Co-Directeur de thèse

---

Institut des Systèmes Intelligents et de Robotique (ISIR),
UMR 7222 Équipe MLIA, 75005, Paris, France
Ecole Doctorale Informatique, Télécommunications et Electronique

Sorbonne University
InstaDeep - BioNTech

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in **Computer Science**

submitted by

VALENTIN MACÉ

DIVERSITY BASED EXPLORATION FOR DEEP REINFORCEMENT LEARNING

---

under the supervision of **Olivier Sigaud** and **Nicolas Perrin-Gilbert**.

Publicly defended on the **18th of October, 2024** in **Paris**, France, to the doctoral committee

Mr. Jean-Baptiste **Mouret**	Inria	Thesis Referee
Mr. Emmanuel **Rachelson**	ISAE-SUPAERO	Thesis Referee
Mr. Cédric **Colas**	Inria & MIT	Thesis Jury
Mr. Stéphane **Doncieux**	Sorbonne Université & CNRS	Thesis Jury
Ms. Sao Mai **Nguyen**	ENSTA Paris	Thesis Jury
Mr. Olivier **Sigaud**	Sorbonne Université	Thesis Supervisor
Mr. Nicolas **Perrin-Gilbert**	Sorbonne Université & CNRS	Thesis Co-Supervisor

---

Institut des Systèmes Intelligents et de Robotique (ISIR),
UMR 7222 Équipe MLIA, 75005, Paris, France
Ecole Doctorale Informatique, Télécommunications et Electronique

Résumé

Cette thèse étudie la famille de méthodes à l'intersection entre l'apprentissage par renforcement profond et les algorithmes évolutionnaires pour résoudre des problèmes où la capacité d'exploration des algorithmes, notamment leur gestion du dilemme exploration-exploitation, joue un rôle crucial. Historiquement, les algorithmes d'apprentissage par renforcement profond sont reconnus comme étant efficients en termes d'efficacité échantillonnale, c'est-à-dire du nombre d'interactions avec l'environnement qu'ils nécessitent pour résoudre un problème d'optimisation, mais ont cependant une capacité limitée à explorer l'espace des solutions possibles, du fait de leurs mécanismes d'exploration simples, souvent basés sur l'ajout de bruit stochastique dans l'espace d'action. D'un autre côté, les méthodes évolutionnaires, et plus particulièrement les algorithmes récents issus de la famille qualité-diversité, présentent une capacité d'exploration supérieure du fait de leurs mécanismes d'exploration axés sur la diversité des solutions au sein d'une population. Ils sont capables de résoudre des problèmes d'optimisation où il est nécessaire d'explorer l'espace des solutions de manière intelligente (en définissant un sous-espace d'intérêt appelé l'espace des comportements), mais sont souvent coûteux en termes d'interactions avec l'environnement.

La première partie des contributions de la thèse (Chapitre 3) se concentre sur l'élaboration d'un algorithme, appelé QD-PG pour "Quality-Diversity-Policy-Gradient", permettant de résoudre des problèmes difficiles d'exploration dans des environnements de contrôle continu (robotique simulée), en se basant sur le cadre algorithmique des méthodes évolutionnaires, et en ayant pour objectif de le rendre efficient grâce aux méthodes basées sur la descente de gradient, issues de l'apprentissage par renforcement. Dans une deuxième partie (Chapitre 4), nous présentons: 1. Un nouvel algorithme de qualité-diversité, appelé MAP-Elites Low-Spread, qui permet de corriger le biais de variance de l'algorithme MAP-Elites et générer des solutions consistantes et régulières dans l'espace des comportements, 2. Une méthode basée sur l'apprentissage profond supervisé permettant de distiller une collection de solutions générées par MAP-Elites Low-Spread dans un réseau de neurones profond unique basé sur l'architecture Transformer, qui est capable de générer des trajectoires conditionnées sur un comportement désiré avec haute précision. Enfin, la dernière partie des contributions (Chapitre 5) introduit un travail en cours, dans lequel nous proposons d'utiliser un modèle basé sur l'architecture Transformer pour prédire l'état final d'automates cellulaires continus à partir de l'état initial et sans connaissance des règles qui les régissent. Nous faisons l'hypothèse qu'un tel modèle peut être employé, entre autres, à détecter automatiquement les patterns intéressants générés par un algorithme de recherche.

Abstract

This thesis is concerned with the family of methods at the intersection between deep reinforcement learning and evolutionary algorithms for solving problems where the algorithms ability to explore, in particular their management of the exploration-exploitation dilemma, plays a crucial role. Historically, deep reinforcement learning algorithms are recognized as being efficient in terms of sample efficiency, but have a limited ability to explore the space of possible solutions, due to their simple exploration mechanisms, often based on the addition of stochastic noise in the action space. On the other hand, evolutionary methods, and more particularly recent algorithms from the quality-diversity family, have a superior ability to explore, due to their exploration mechanisms based on the diversity of solutions within a population. They are capable of solving optimization problems where it is necessary to explore the solution space intelligently (by defining a subspace of interest called the behavior space), but are often costly in terms of interactions with the environment.

The first part of the contributions of this thesis (Chapter 3) focuses on the development of an hybrid algorithm, called Quality-Diversity-Policy-Gradient (QD-PG), for solving difficult exploration problems in continuous control environments (simulated robotics), based on the algorithmic framework of quality-diversity methods, and aiming to make it sample efficient using gradient-based methods derived from reinforcement learning. In the second part (Chapter 4), we present: 1. A new quality-diversity algorithm, called MAP-Elites Low-Spread, which corrects the variance bias of the MAP-Elites algorithm and generates constant and regular solutions in the behavior space, 2. A method based on supervised deep learning for distilling a collection of solutions generated by MAP-Elites Low-Spread into a single deep neural network based on the Transformer architecture, which is capable of generating trajectories conditioned on a desired behavior with high accuracy. Finally, the last contribution part (Chapter 5) introduces a work in progress, where we propose using a Transformer-based model to predict the final state of continuous cellular automata from an initial state, without prior knowledge of their governing rules. We hypothesize that this model could be employed to automatically detect interesting patterns generated by a search algorithm.

Contents

Chapter 1

Introduction

Any sufficiently advanced technology
is indistinguishable from magic.

Arthur C. Clarke.

In this chapter, we offer a brief and informal overview of the concepts and personal motivations driving this thesis. We discuss deep learning, evolutionary algorithms, and reinforcement learning, as well as the problems of interest we seek to tackle. Finally, we outline the structure and key contributions developed in subsequent chapters.

Contents

1.1 Motivation

What makes it possible for a piece of refined rock to distinguish an image of a pedestrian from that of a road? What makes it possible for a piece of refined rock to make the decision to brake a car that is about to hit a pedestrian?

Starting from logic circuits implemented — to this day — in silicon, to complex applications, computer science and computer engineering can be seen as the art of building layers of abstractions to solve problems increasingly akin to human tasks. As the abstractions grow and continue to improve, computers provide tools to formulate and tackle problems that were previously out of reach. Video games, weather forecasts, special effects, business and medical software, videoconferencing, are just a few of the applications made possible by high-level abstractions built on the deep and robust stack of software and hardware. For instance, creating a state-of-the-art video game without relying on high-level programming tools (such as a video game engine or simply a high-level programming language) would be, if not impossible, at least unrealistic, although the hardware technically allows it.

There is however an abstraction limit to what classical software (as opposed to AI-based software) can achieve. Some of the most elaborated classical software tools enable, for example, the existence of high-security financial systems used worldwide, complex physics simulations to model the universe from very large to very small scales (Bertschinger, 1998; Feynman, 2018), the instantaneous connection of people who live far apart — among many other things — but there is no piece of classical software that can, with human or superhuman accuracy, take a picture and describe its content; read a book and summarize it; observe the environment surrounding a car and make sequential decisions to reach the destination safely. In other words, classical software is unable to incorporate and deal with high-level semantic concepts that are close to what humans do. It is unable to handle semantic ambiguity with much success, as it relies on rigid rules that are not learned and that are not suited to grasp the complexity of the world. To return to the universe simulation example, suppose we obtain a massive amount of data after running it and, for the sake of argument, that we are looking for very particular galaxy patterns among the billions of structures generated, such as something resembling a human face. It would be incredibly difficult, if not impossible, to program classical software to iterate over the data and automatically detect such a pattern. However, it would be fairly easy to explain the task to other humans, who in turn have their own limitations — most notably speed and available brain time. Pushing the argument further, if we are now investigating dark matter in the universe and looking for a very specific type of galaxy pattern that may or may not exist, neither classical software nor humans would be of much help in such a situation, because the semantic concepts that we are investigating are beyond the reach of rigid computation rules, and beyond what is intuitive for the human mind. These simple examples underline the need

for more advanced techniques (or abstractions layers) closing the gap between computers and real-world complexity.

Machine learning (ML) — its past successes and aspirations — is arguably the best candidate to bridge this gap, enabling computers to handle tasks previously thought to require human intelligence. The main difference between classical software and AI-based software (or machine-learning-based software) lies in the fact that the latter is learning-driven, and sees its behavior defined by a learning phase based on data or interactions with an environment. Machine Learning can be seen as the ultimate layer of abstraction, building on top of classical software and allowing computing systems to handle semantic ambiguity across various fields, sometimes even better than the most skilled humans. In 2012, during the ImageNet Large Scale Visual Recognition Challenge (Russakovsky, Deng, Su, et al., 2015), for the first time, a convolutional neural network (CNN) outperformed all other classical systems in an image recognition competition (Krizhevsky, Sutskever, and Hinton, 2012), marking a turning point in the use of CNNs for computer vision. In 2016, AlphaGo (Silver, Huang, Maddison, et al., 2016), an agent trained with deep reinforcement learning, defeated world Go champion Lee Sedol in a five-game series, winning 4-1. This victory demonstrated the ability of machine learning algorithms to master complex strategic decision making, and even to demonstrate surprising strategic creativity, with the now famous move 37. In 2023, ChatGPT, a conversational agent based on the GPT-4 language model (Achiam, Adler, Agarwal, et al., 2023), has shown exceptional competence in natural language conversations, surpassing previous systems in terms of fluency, relevance and consistency. Applications for GPT-4 and ChatGPT include content writing, question answering, programming assistance, thesis writing[1], and many other tasks requiring a high level of natural language understanding. Perhaps more importantly, recent work has demonstrated the ability of machine learning systems to understand semantic concepts where human intuition falls short. In the field of biology, AlphaFold (Jumper, Evans, Pritzel, et al., 2021; Abramson, Adler, Dunger, et al., 2024) has shown a remarkable understanding of the protein space, allowing the prediction of three-dimensional proteins from their amino acid sequences with unprecedented accuracy. Regarding these past accomplishments and future hopes, computer science may be the closest thing we have to magic — and computer scientists, to wizards.

Machine learning is a vast field that includes a number of historically important sub-domains. In this thesis, we essentially build on three of them, which have seen immense progress in the last decade. First, deep learning (supervised or self-supervised) forms the backbone of many modern AI systems, leveraging large datasets to train models that are suited for various tasks such as CNNs (LeCun, Bottou, Bengio, et al., 1998) for vision, or Transformers (Vaswani, Shazeer, Parmar, et al., 2017) for natural language processing. Second, evolutionary algorithms (and more specifically quality-diversity algorithms) are a class of optimization techniques

[1]This is a joke.

inspired by the process of natural evolution, allowing for the exploration of a diverse set of solutions and promoting robustness and adaptability in AI systems (Cully, Clune, Tarapore, et al., 2015). Third, reinforcement learning (RL) is an optimization framework inspired by the way animals and humans learn through interactions with their environment, aiming to maximize cumulative rewards or minimize punishments. Recently, RL has found applications in a variety of fields such as plasma control for nuclear fusion (Degrave, Felici, Buchli, et al., 2022) or autonomous navigation of stratospheric balloons (Bellemare, Candido, Castro, et al., 2020). In this thesis, we focus our research efforts on methods at the intersection between deep neural networks, reinforcement learning and evolutionary algorithms for controlling robots in simulated environments. We investigate new algorithms that mitigate the respective drawbacks of aforementioned methods, and we aim to tackle problems that cannot be solved with traditional evolutionary algorithms or reinforcement learning because they require a tailored approach to the exploration-exploitation dilemma; we call them *hard-exploration problems*.

1.2 Deep Learning

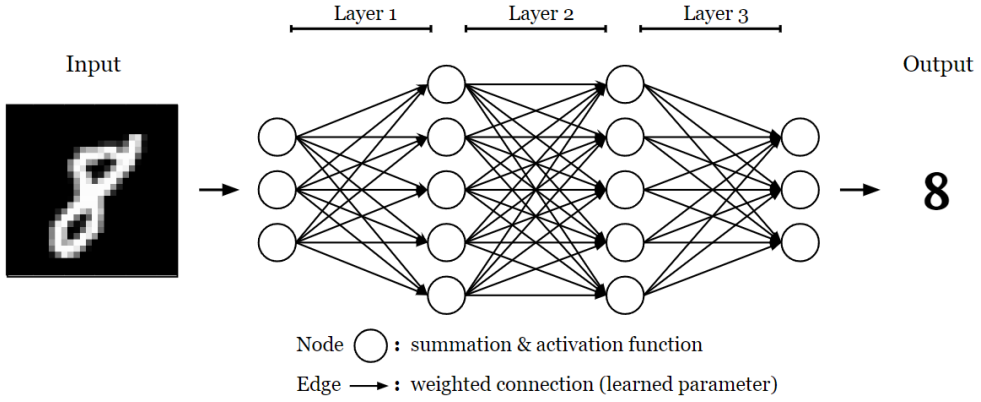


Figure 1.1 – High-level view of a trained multilayer perceptron neural network. An input image representing the hand-written digit "8" is fed to a trained neural network, which is able to identify it correctly.

Deep learning is the backbone of the post-2010 AI revolution, a transformative period that arguably continues to this day. Historically, deep learning is (bio)inspired by the idea that a network of relatively simple atomic structures (artificial neurons) can efficiently abstract and model complex data. The very first scientific mention of a neural network dates back to 1943 when a neurophysiologist and a logician introduced a mathematical model of neural networks based on the functioning of the human brain (McCulloch and Pitts, 1943) . They demonstrated that a network of these simple units, or neurons, could perform any logical function, and also introduced the concept of weights (synaptic strengths) and thresholds (activation levels),

which are fundamental components of modern neural networks (see Figure 1.1). The term "deep" in "deep learning" is attributed to the increased depth, or number of layers, in artificial neural networks compared to earlier models.

The general framework of deep learning in the context of supervised learning is fairly easy to understand: a neural network takes an input (usually an array of real numbers) which represents an image, a sound, a word or any data source that can be represented in numbers, and performs a sequence of operations to deliver a verdict (the output). Figure 1.1 represents a simple, fully connected feed forward neural network that takes the image of an hand-written digit and recognizes it successfully. Each node of the network performs a basic weighted summation of its inputs followed by an activation function[2]; these weights typically constitute the network learnable parameters. Of course, in the beginning, and without any training, there is no reason for a network of interconnected artificial neurons to produce better results than pure randomness, the network predictions are random and of no interest. The power of neural networks lies in the fact that they feature a large number of parameters that can be adjusted to converge towards a goal. During the training procedure for a given task, which can range from image classification to stock value prediction, the neural network — or model — is fed with a large amount of labeled examples and asked to make predictions about these examples. Then, these predictions are compared with ground truth labels (hence the name "supervised learning") and the accumulated error of the model is measured. Finally, thanks to mathematical tools such as automatic differentiation, the parameters of the network are adjusted so as to minimize the measured error and the procedure is repeated until convergence.

Remark 1.1 (Ground truth labels)**.** *In the context of supervised learning, each training example has a ground truth label associated, which is often predetermined by a human. Other training regimes exist, such as self supervised learning, where labels are collected automatically. It is important to note that this difference does not affect the learning algorithm itself but only the data collection part.*

In practice, neural network architectures have evolved considerably, becoming more complex and adapted to the task on which they are trained. For instance, vision-based systems possess features capable of capturing translational invariances within an image — a cat in the top-right corner of an image should be just as well recognized and detected as a cat in the bottom-left corner —(Krizhevsky, Sutskever, and Hinton, 2012), while language models are built to pay more or less attention to the different words in a sentence (Vaswani, Shazeer, Parmar, et al., 2017). Generally, one instance of a trained neural network corresponds to one task[3]: a model that has been trained to classify images is unable and unsuited to do other tasks. It is unable

[2]The activation function helps a neuron decide whether to pass a signal forward, enabling neural networks to learn and recognize complex patterns in the data.

[3]This limitation tends to disappear with the arrival of large multitask and multimodal models.

because its parameters have been trained to perform image classification on a given domain, and unsuited because its architecture (mainly its input and output layers, which are usually rigid) are specifically shaped for this task.

Deep neural networks are now everywhere, transforming industries and enhancing daily life in remarkable ways. In the automotive industry, companies use deep learning for autonomous driving, enabling cars to navigate and make decisions with minimal human intervention (Bojarski, Del Testa, Dworakowski, et al., 2016; Bachute and Subhedar, 2021), in finance, they are used for fraud detection, analyzing vast amounts of transaction data to identify unusual patterns and prevent fraudulent activities (Jurgovsky, Granitzer, Ziegler, et al., 2018), in healthcare, deep learning helps medical diagnosis with models that can predict breast cancer risk from mammograms (McKinney, Sieniek, Godbole, et al., 2020), etc. Interestingly, neural networks have more to offer than supervised learning applications. Deep neural networks, which have been proven to be universal function approximators (Hornik, Stinchcombe, and White, 1989), are also useful when used as representation tools in other ML frameworks, such as evolutionary algorithms and reinforcement learning.

1.3 Evolutionary Algorithms

Evolutionary algorithms (EAs) are another area of modern artificial intelligence, inspired by the principles of natural selection and genetics. These algorithms provide robust optimization techniques that can solve optimization problems by evolving solutions over time. The concept of evolutionary computation dates back to the 1960s, when pioneers began exploring the potential of mimicking biological evolution to optimize mathematical functions and solve engineering problems (Holland, 1992; Fogel, 1998). Some EAs notable successes can be found across various fields, for instance, they have been employed in the aerospace industry to design innovative antenna structures for satellites, resulting in more efficient and effective designs compared to traditional methods (Hornby, Globus, Linden, et al., 2006), these algorithms have also been used for design optimization to solve real-world aerodynamic problems (Lian, Oyama, and Liou, 2010), and in robotics, EAs have been used to discover robust and efficient gaits for articulated robots (Cully, Clune, Tarapore, et al., 2015).

Remark 1.2 (EAs and deep learning)**.** *Evolutionary algorithms are not alternatives or competitors to deep learning; they solve problems that are different from what deep learning does, and sometimes, they even use deep neural networks as tools for solution encoding. In this sense, they constitute an optimization framework that can be compared to other optimization frameworks, such as reinforcement learning* (*Chalumeau, Boige, Lim, et al., 2022*).

The evolutionary optimization framework generally involves the evolution of a population of potential solutions to a given problem. These solutions, often represented as arrays of numbers or other data structures, undergo processes analogous to genetic operations such as selection, crossover (recombination), and mutation. Initially, at the beginning of an evolutionary algorithm, a population of solutions is randomly generated, then: 1. Each individual in this population is evaluated against the problem at hand using a fitness function, which quantifies how well it performs. 2. Once evaluated, a selection mechanism filters solutions by prioritizing the survival of the fittest individuals, thereby ensuring that better solutions are more likely to pass their traits to the next generation. 3. Finally, some of the selected solutions are mutated and/or recombined to create the next generation (population) of solutions. Recombination (crossover) operations combine parts of two or more selected individuals to produce offspring, potentially inheriting the strengths of both parents. Mutation introduces random changes to individual solutions, promoting diversity and enabling the exploration of the solution space. This process of evaluation (1), selection (2), mutation and crossover (3) goes on repeatedly (as illustrated in Figure 1.2) until a termination criterion is met. For example, one could use EAs to search for an efficient, gas-saving airfoil with low air resistance using evolutionary algorithms, starting with a population of random airfoil designs defined by a set of numbers capturing their shape, then iteratively testing them in a flying simulator to determine their fitness and applying selection, mutation and crossover operations until a sufficiently good airfoil is found.

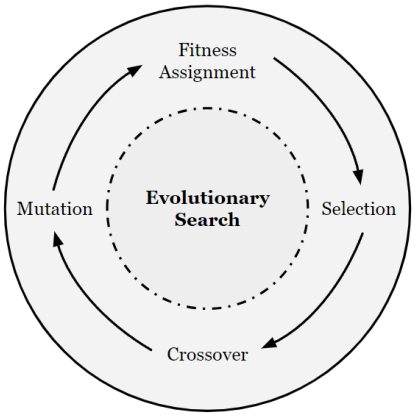


Figure 1.2 – Diagram of a typical evolutionary algorithm.

Evolutionary algorithms are useful tools for optimization, they are generally considered robust — meaning they are not extremely sensitive to the parameters that determine their functioning —, and said to be black-box optimization algorithms, meaning that they do not require prior knowledge on the problem at hand: they simply need access to an estimated fitness of current solutions through evaluation. However, classical EAs typically suffer from low efficiency: they usually require numerous iterations, and thus many fitness evaluation, to get satisfactory solutions. Depending on the problem under consideration, access to fitness evaluation can be computationally expensive. Continuing with the example of airfoil design optimization, access to an accurate simulator can be difficult, the cost of simulation itself can be high, and it is not possible to envisage testing in the real world for obvious reasons.

In addition to this drawback, classical EAs are not effective in solving hard-exploration problems (Lehman and Stanley, 2011a). They are designed to output a single solution to any given problem and rely on simple exploration mechanisms (mainly mutations) to promote diversity and explore the space of all possible solutions, which are often insufficient in hard-

exploration problems. In other words, classical EAs would not have been able to invent the Fosbury flop to solve the discipline of high jumping. In this thesis, we will focus our efforts on quality-diversity (QD) algorithms, which are a (more recent) subclass of evolutionary algorithms aiming a producing a collection of diverse solutions (instead of just one) (Cully and Demiris, 2017). The structural diversity-seeking mechanisms in QD algorithms allows the effective exploration of the solution space, often enabling these algorithms to succeed where conventional EAs fail (Chalumeau, Pierrot, Macé, et al., 2022).

1.4 Reinforcement Learning

Reinforcement learning (RL) is a field within machine learning that focuses on training autonomous agents to take actions in an environment to maximize cumulative reward. RL is inspired by the behavioral learning processes observed in nature, where organisms learn to adapt their actions based on the consequences they experience. This approach involves agents exploring their environment, exploiting their knowledge, and iteratively improving their decision-making process. As opposed to natural evolution, where changes in organisms are mainly observed at the species level, reinforcement occurs during the lifetime of an organism, and modifies its decision making process based on its life experience. Early contributions such as the law of effect (Thorndike, 1898, 1927), established through study of animal intelligence, and the operant conditioning framework (Skinner, 1938), which emphasizes the role of reinforcement and punishment in shaping behavior, laid the foundations for the development of modern RL.

The theoretical underpinnings of RL are rooted in the work of Richard Sutton and Andrew Barto, who established the fundamental RL concepts of agent, policy, reward, and environment interaction (Sutton and Barto, 1998). Figure 1.3 illustrates the fundamental concepts of reinforcement learning with a familiar scene: An agent (the child) interacts with an environment (the videogame) by observing the TV screen, which provides information on the current state of the game, and by taking actions that will have an impact on the course of the game. The observation-action loop goes on and, sometimes, at a frequency that depends on the environment, a feedback (called the reward) is given to the agent. In the example of Figure 1.3, the feedback could be a score that increments or decrements in real-time, providing information about the agent's performance at each moment of the game. Generally, the role of reinforcement learning algorithms is to optimize the se-

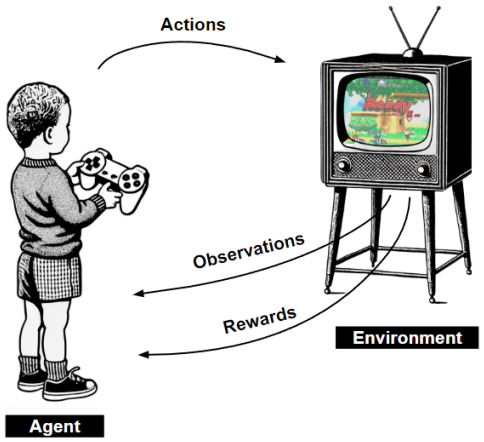


Figure 1.3 – Reinforcement learning fundamental concepts.

quential decision-making of the agent (called the policy) in order to maximize the cumulative reward obtained (i.e., be good at the task at hand). In other words, the goal of an RL algorithm is to generate a policy — a set of rules that determines what actions the agent should take based on observations — that maximizes rewards when applied to a given task. To do so, an RL algorithm uses the data generated by agent-environment interactions, analyzes which actions (or series of actions) led to positive outcomes (rewards), and which actions led to negative outcomes. Finally, the policy of the agent is modified to reinforce valuable actions and penalize detrimental ones. This process of data collection via agent-environment interactions and policy adjustment is repeated until sufficient performance is achieved.

As with evolutionary methods, reinforcement learning has benefited from the use of deep neural networks to represent policies[4] — leading to the emergence of the term "deep reinforcement learning". Deep RL has achieved significant successes across a variety of fields. Notably, it has been used to train autonomous agents that excel in complex games (Silver, Hubert, Schrittwieser, et al., 2017; Vinyals, Babuschkin, Czarnecki, et al., 2019), perform object manipulation tasks with high precision (Levine, Finn, Darrell, et al., 2016), or even optimize the cooling systems of data centers to reduce energy consumption (Evans, Gao, Anderson, et al., 2018). RL algorithms are known to be efficient, meaning that they usually require less interactions with the environment than evolutionary algorithms (Pourchot and Sigaud, 2018). However, perhaps even more so than evolutionary methods, pure RL algorithms struggle with hard-exploration problems. They also are less scalable and notoriously more difficult to train than evolutionary algorithms, being more sensitive to slight variations of their parameters (Chalumeau, Boige, Lim, et al., 2022).

Remark 1.3 (Policy, solution, agent ...)**.** *There is a strong semantic proximity between the terms "policy" (as used in RL) and "solution" (as used in evolutionary methods). Indeed, when both families of algorithms use neural networks to solve sequential decision-making problems, these two terms can be used almost interchangeably. The "agent" (as used in RL) is also close to these two concepts in this context, with the difference that the agent is the broader entity that encompasses the learning and decision-making processes, while the policy is a specific part of the agent that determines its actions. In this work, we will sometimes use these terms interchangeably, depending on the context.*

1.5 Hard-Exploration Problems

Although we focus on simulated robotic environments in this thesis, hard-exploration problems exist in many areas of our world and are difficult, if not impossible, to solve with traditional

[4]Policies in RL can be anything that select actions (deterministically or not) given observations: a policy can be implemented with a set of explicit rules, as well as with a neural network whose operation is opaque to us.

optimization methods. A famous example of such a problem can be found in the sport of high jumping. Historically, athletes used jumping techniques such as the scissors jump or the straddle, which are arguably the most intuitive jumping techniques for a human attempting to leap over an obstacle. These techniques dominated the discipline for decades, and to many athletes and sport experts, it was inconceivable that a drastically different, counterintuitive, and more effective move existed. However in 1968 during the Mexico City Olympics, Dick Fosbury introduced a revolutionary technique called the Fosbury flop, which involves jumping with the back to the bar and arching over it with the back facing down. This new move quickly became the dominant method in high jump, allowing Fosbury to win the gold medal and enabling other athletes to clear old records by a significant margin. In essence, the history of high jumping can be thought of as an optimization process, where athletes trained for decades to marginally improve over previous records by slightly adapting their jumping technique and muscular capacities. As with many hard-exploration problem, the discovery of the best solution (to date) did not come as a marginal improvement over the best existing ones; it required the exploration of counterintuitive options (Yin, Yang, Van De Panne, et al., 2021).

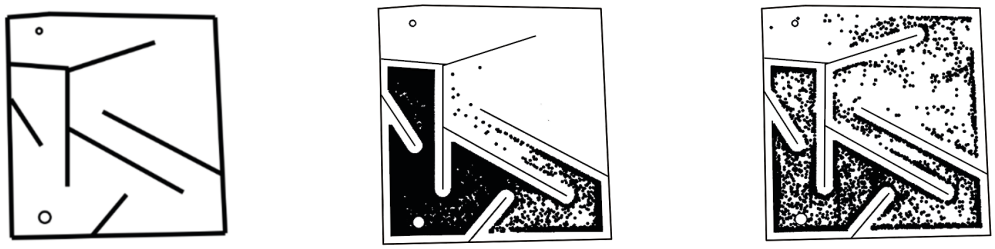


Figure 1.4 – A canonical hard-exploration problem. From left to right: 1. Picture of the maze where the agent (larger circle) is rewarded the closer it gets to the exit (smaller circle), 2. Results of using a classical evolutionary algorithm to solve the problem, most tries (black dots) end up in dead ends, 3. Results of running an exploration-oriented algorithm, which solves the maze by focusing on exploration and novelty search. Taken from Lehman and Stanley (2011a).

In the realm of computers, hard-exploration problems can appear in many different forms. A canonical example of such a problem is that of the maze depicted in Figure 1.4. Here, the agent (the larger circle) starts in the bottom-left corner of the maze, takes an action at each timestep to slightly move in any chosen direction, and is rewarded in proportion to its proximity to the exit (the smaller circle) in the upper-left corner — the closer the better. Naturally, its goal is to get as close to the exit as quickly as possible to maximize the cumulative reward obtained. In this example, the agent has no way of knowing there are walls separating it from the exit, the only information available being its current position in the maze and the reward obtained after each action taken. In such a maze, a naive agent that optimizes for rewards greedily — thus displaying insufficient exploration of the environment — will most likely end up running upward, in the dead-end trap formed by the walls, because it is the fastest way to obtain high rewards quickly. However, even though this strategy quickly decrease the distance between the

agent and the exit, it cannot lead to the resolution of the maze because of the blocking walls; the global solution involves first moving away from the exit to get around the walls, and finally reaching the exit. Middle picture of Figure 1.4 illustrates the results of a classical evolutionary algorithm on this problem, each attempt to reach the exit from the beginning is represented as a black dot which indicates the final position of the agent. As expected, all attempts to reach the exit fail, and most fall into the trap of blindly maximizing the reward by running towards the walls.

For most classical evolutionary algorithms, exploration capabilities rely on random mutations of the solutions in the population. As their name suggests, these mutations are random and allow *a certain degree* of exploration by diversifying the behavior of solutions. However, these mutations are often insufficient to generate a wide variety of original behaviors, and when an original (but necessarily suboptimal) behavior appears — a behavior that could serve as a stepping stone for next generations —, it is often discarded by the selection step, which filters suboptimal solutions. In reinforcement learning, exploration is ensured by adding random noise to the actions chosen by the policy during agent-environment interactions for data collection. This random noise allows to explore different actions and to not always choose the one that is considered optimal. Yet, exploration in the action space alone is often insufficient in complex problems (Amin, Gomrokchi, Satija, et al., 2021) and reinforcement learning is generally regarded as less effective in hard-exploration problems than its evolutionary counterparts (Colas, Sigaud, and Oudeyer, 2018).

While exploration is necessary as a tool for optimization, exploration in itself is an interesting subject of study and is often associated with robustness, surprise, adaptability and serendipity (Cully, Clune, Tarapore, et al., 2015; Mouret and Clune, 2015). Instead of searching for the most effective way to accomplish a task, one may want to cover the space of possible ways to accomplish it; to generate as many different skills without concern for their performance, sometimes leading to unanticipated solutions (Eysenbach, Gupta, Ibarz, et al., 2018; Lehman, Clune, Misevic, et al., 2020). The right part in Figure 1.4 depicts the results of Novelty Search (NS), an exploration-oriented evolutionary algorithm that focuses on producing solutions different from those already generated without taking performance into consideration, thereby maximizing diversity within the population of solutions (Lehman and Stanley, 2011a). By completely ignoring the objective — the exit of the maze and the necessity for the agent to reach it — and focusing solely on exploration, NS is able to solve the problem and find the exit, whereas goal-oriented algorithms fail to find it. Following NS, numerous works have highlighted the benefits of algorithms that dedicate an important part of their operation to exploration, leading among other things to the emergence of the field of quality-diversity (Chatzilygeroudis, Cully, Vassiliades, et al., 2021).

1.6 Outline and Contributions

Summary

- Chapter 2 introduces the technical and theoretical background of the most important concepts (for this thesis) in supervised learning, evolutionary algorithms and reinforcement learning. We also briefly discuss the software engineering challenges related to the implementation of our contributions.
- Chapter 3 presents QD-PG (Pierrot, Macé, Chalumeau, et al., 2022), a hybrid quality-diversity reinforcement learning algorithm that solves challenging exploration problems in a sample efficient manner by leveraging reinforcement learning policy gradient methods for diversity search.
- Chapter 4 presents the Quality-Diversity Transformer (Macé, Boige, Chalumeau, et al., 2023), a unique policy that learns from all the diverse policies produced by a quality-diversity algorithm, and that is able to reproduce their respective behavior on demand, with high accuracy and some degree of generalization. This chapter also proposes a new method to solve the reproducibility (high-variance) problem of quality-diversity policies in uncertain environments.
- Chapter 5 is about a work in progress on the automatic search of singular, diverse and interesting patterns in continuous cellular automata. We provide early results using a Transformer-based model to predict continuous cellular automata outcomes and briefly discuss a method to use this model as the basis of a search algorithm.
- Chapter 6 briefly summarizes the contributions and results of this thesis but also provides additional practical details, and a personal and critical point of view on the presented contributions.

List of publications

First author publications in international conferences with proceedings

- Valentin Macé, Raphaël Boige, Felix Chalumeau, Thomas Pierrot, Guillaume Richard, and Nicolas Perrin-Gilbert (2023). The quality-diversity transformer: Generating behavior-conditioned trajectories with decision transformers. **In** *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1221–1229. Chapter 4 mainly repeats this article.
- Thomas Pierrot[*], Valentin Macé[*], Felix Chalumeau, Arthur Flajolet, Geoffrey Cideron, Karim Beguir, Antoine Cully, Olivier Sigaud, and Nicolas Perrin-Gilbert (2022). Diversity policy gradient for sample efficient quality-diversity optimization. **In** *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1075–1083. Chapter 3 mainly repeats this article.

Collaborations in international conferences or journals with proceedings

- Felix Chalumeau, Raphael Boige, Bryan Lim, Valentin Macé, Maxime Allard, Arthur Flajolet, Antoine Cully, and Thomas Pierrot (2022). Neuroevolution is a competitive alternative to reinforcement learning for skill discovery. *arXiv preprint arXiv:2210.03516*.
- Felix Chalumeau, Bryan Lim, Raphael Boige, Maxime Allard, Luca Grillotti, Manon Flageat, Valentin Macé, Guillaume Richard, Arthur Flajolet, Thomas Pierrot, et al. (2024). QDax: A library for quality-diversity and population-based algorithms with hardware acceleration. *Journal of Machine Learning Research* 25.108, pp. 1–16. Section 2.4 introduces the QDax framework.

Workshop collaborations in international conferences or preprints

- Felix Chalumeau, Thomas Pierrot, Valentin Macé, Arthur Flajolet, Karim Beguir, Antoine Cully, and Nicolas Perrin-Gilbert (2022). Assessing Quality-Diversity Neuro-Evolution Algorithms Performance in Hard Exploration Problems. *arXiv preprint arXiv:2211.13742*.
- Thomas Pierrot, Valentin Macé, Jean-Baptiste Sevestre, Louis Monier, Alexandre Laterre, Nicolas Perrin, Karim Beguir, and Olivier Sigaud (2021). Factored action spaces in deep reinforcement learning.

Software contributions

github.com/adaptive-intelligent-robotics/QDax

[*] Authors contributed equally.

Collaborations conducted before this thesis

- Valentin Macé and Christophe Servan (2019). Using Whole Document Context in Neural Machine Translation. **In** *Proceedings of the 16th International Conference on Spoken Language Translation*. Hong Kong: Association for Computational Linguistics.
- Bernard Espinasse, Sébastien Fournier, Adrian Chifu, Gaël Guibon, René Azcurra, and Valentin Mace (2019). On the Use of Dependencies in Relation Classification of Text with Deep Learning. **In** *International Conference on Computational Linguistics and Intelligent Text Processing*. Springer, pp. 379–391.

Chapter 2

Background

I asked myself childish questions
and proceeded to answer them.

Albert Einstein.

This chapter outlines the technical background of the concepts introduced in Chapter 1. Starting with deep learning and neural networks, we then explore quality-diversity methods as a subset of evolutionary algorithms for enhanced exploration, and lastly, we discuss the framework of reinforcement learning. Finally we present the tools that enable training neural networks.

Contents

2.1 Deep Learning and Neural Networks

In what follows, we present the base components of deep learning and neural networks. We start with the introduction of the multilayer perceptron (Goodfellow, Bengio, and Courville, 2016), which is the main neural network architecture used throughout this thesis, and then present the Transformer (Vaswani, Shazeer, Parmar, et al., 2017), a groundbreaking architecture born of the field of neural machine translation that is now ubiquitous in machine learning.

2.1.1 Multilayer Perceptrons

Multilayer Perceptrons (MLPs) are one of the earliest and most well-known classes of feed-forward neural networks. Historically, MLPs have been fundamental in the development of neural network research due to their simplicity and versatility. They have played a crucial role in supervised learning tasks (classification and regression) although today they tend to be abandoned in favor of more specific and complex architectures such as CNNs, Transformers or other hybrid models. An MLP typically consists of one input layer, one or more hidden layers, and one output layer. Each layer is composed of computational nodes (also called neurons) that are fully connected to nodes in the next layer. Figure 1.1 provides a high-level representation of an MLP.

1. **Input Layer**: The input layer receives the input features of the data. Each neuron in the input layer represents an input feature.

2. **Hidden Layers**: The hidden layers perform computations and transformations on the inputs received from the previous layer. Each neuron in a hidden layer applies a weighted sum of the inputs followed by a non-linear activation function.

3. **Output Layer**: The output layer produces the final predictions. The number of neurons in the output layer corresponds to the number of target classes (for classification) or the number of output values (for regression).

The operation of an MLP can be described mathematically as follows: Consider an MLP with one hidden layer. Let $\mathbf{x}$ be the input vector, $\mathbf{W}^{(1)}$ and $\mathbf{b}^{(1)}$ be the weights and biases of the hidden layer, the output of the hidden layer can be formulated as:

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$
$$\mathbf{a}^{(1)} = f(\mathbf{z}^{(1)}).$$

Here, $\mathbf{z}^{(1)}$ is the linear combination of inputs, and $\mathbf{a}^{(1)}$ is the activation of the hidden layer neurons. The function f is the activation function introducing non-linearity into the model, allowing it to learn complex patterns. Figure 2.1 illustrates the functioning of a single MLP

neuron (left) and plots the rectified linear unit (ReLU) activation function (Nair and Hinton, 2010) (right), which is the activation function mainly used in the remainder of this thesis.

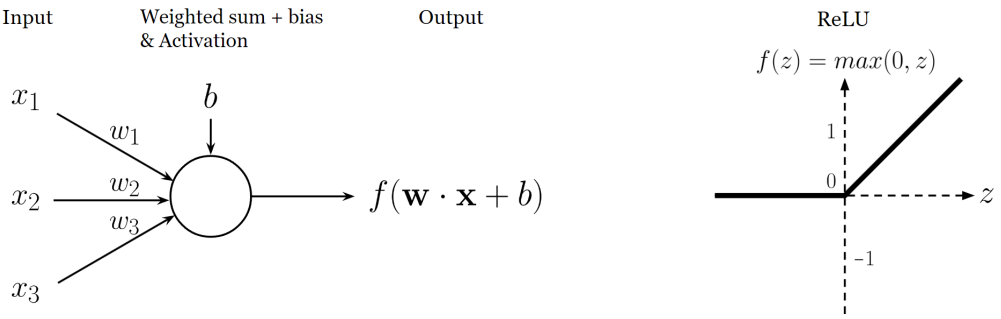


Figure 2.1 – Left: A single MLP node computes its output as a weighted sum of the input (plus a bias that acts as an additional parameter) and passes the result through a nonlinear activation function. Right: Example of a nonlinear activation function, the rectified linear unit (ReLU).

Similarly to the hidden layer, the output layer computation can be formulated as:

$$\mathbf{z}^{(2)} = \mathbf{W}^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)}$$
$$\mathbf{y} = g(\mathbf{z}^{(2)}),$$

where $\mathbf{a}^{(1)}$ is the activation of the previous (hidden) layer, and $\mathbf{W}^{(2)}$ and $\mathbf{b}^{(2)}$ are the weights and biases of the output layer. Here, $\mathbf{z}^{(2)}$ is the linear combination of hidden layer activations, and $\mathbf{y}$ is the output of the MLP. The function g is the activation function for the output layer, which varies depending on the task (e.g., softmax for classification or identity for regression).

2.1.2 The Transformer

The Transformer (Vaswani, Shazeer, Parmar, et al., 2017) is a popular model architecture that was originally designed for natural language processing (NLP) tasks such as language translation (Devlin, Chang, Lee, et al., 2018), but has since been applied to a variety of other tasks, including image and speech recognition (Dong, Xu, and Xu, 2018; Dosovitskiy, Beyer, Kolesnikov, et al., 2020), text generation (Brown, Mann, Ryder, et al., 2020) and sequence modeling for RL (Chen, Lu, Rajeswaran, et al., 2021). Unlike older architectures that are used to process sequential data, such as recurrent neural networks (RNNs) or long short-term memory networks (LSTMs), Transformers rely on self-attention mechanisms to process input sequences, enabling more efficient parallelization and improved performance. RNNs and LSTMs process sequential data by maintaining a hidden state that is updated at each step, inherently limiting their ability to parallelize computations and making them prone to errors over long sequences. In contrast, Transformers use self-attention to directly model

dependencies between all elements of a sequence at once, allowing for better handling of long-range dependencies and more efficient training.

The Transformer historically consists of an encoder-decoder architecture, with both the encoder and decoder composed of multiple layers. However, later works show that encoder-only (resp. decoder-only) architectures are also very effective (Devlin, Chang, Lee, et al., 2018; Chowdhery, Narang, Devlin, et al., 2023). For the sake of simplicity, and because we exclusively use decoder-only models in this work — to be precise, the GPT architecture (Radford, Narasimhan, Salimans, et al., 2018) —, let us introduce the Transformer through the lens of a decoder-only architecture. The decoder-only Transformer keeps the core components of the original one, but focuses on the decoding part. This design allows the model to predict the next element (or token) in a sequence, making it particularly effective for autoregressive tasks, i.e., tasks that involve generating sequences one element at a time based on previously generated elements (text completion, for example). A single decoder is usually composed of a stack of identical layers, each of these identical layers contain three sub-layers:

1. **Multi-Head Self-Attention**: This sub-layer allows the model to pay *attention* to different parts of the input sequence simultaneously. It computes attention scores for each token in the sequence relative to every other token, enabling the model to capture contextual relationships.

2. **Feed-Forward Neural Network**: Following the multi-head self-attention, this sub-layer consists of a fully connected feed-forward network (similar to an MLP output layer) applied to each position independently and identically.

3. **Layer Normalization and Residual Connections**: Each aforementioned sub-layer is followed by layer normalization, which addresses the internal covariate shift problem (Ba, Kiros, and Hinton, 2016), and residual connections to ensure stable training and better gradient flow (see Section 2.1.3).

At the heart of the Transformer architecture is the self-attention mechanism (1), which computes a weighted sum of input values — the weights being determined by the similarity between query and key pairs. Self-attention can be described as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.1}$$

where Q (queries), K (keys) and V (values) are matrices derived from the inputs, and d_k is the dimension of the key vectors. Put simply, to encode a given token x, the attention mechanism works in three steps: 1. It associates three vectors with each input token (a query vector, a key vector, and a value vector), 2. The query vector for x is used to compute scores using the key vectors of every other tokens in the sequence. These scores are computed as the dot product of the two vectors and capture how much focus to other parts of the sequence is needed to

encode x (i.e., taking into account the context) and 3. It multiplies these scores (softmax values) with their respective value vector and sum up the weighted value vectors, which produces the encoding vector for x. Since this process is not intrinsically sequential, it can be carried out for all input tokens in parallel as depicted in Equation 2.1 where queries, keys and values for are aggregated into matrices (Q, K and V respectively).

In the context of a GPT-like, decoder-only architecture, multi-head self-attention is made *causal* by ensuring the prediction for each position depends only on previous inputs. In other words, when processing a token at position t, the model can only attend to tokens anterior to t, preventing leftward information flow in the decoder and preserving the auto-regressive property. To do so, a causal mask is added to Equation 2.1 as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V \tag{2.2}$$

where M is the mask that sets the attention scores to $-\infty$ for future positions to ensure causality.

Instead of performing attention a single time, the Transformer makes use of multi-head attention, allowing the model to jointly attend to information from different representations at different positions. Multi-head attention works as follows:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O,$$

where each head is computed as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V),$$

and where W_i^Q, W_i^K, and W_i^V are projection matrices for the i-th head (i.e., learnable matrices that distinguishes one head from another). W^O is the output projection matrix.

Finally, the position-wise feed-forward neural network (2) that follows the masked multi-head attention mechanism can be described as:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2,$$

where W_1 and W_2 are weight matrices, and b_1 and b_2 are bias vectors.

Remark 2.1 (Neural network architectures)**.** *There is a limited analogy between biological neural networks and artificial ones, which tends to fade increasingly over time. Artificial neural networks really are just a succession of simple, (hopefully) differentiable operations: the choice of their architecture is thus free, and nothing fundamentally changes between the training of a*

> *small MLP and that of a large Transformer. See "Deep Learning est mort. Vive Differentiable Programming!" (LeCun, 2018).*

2.1.3 Training Neural Networks

Neural networks are parameterized functions that can be tuned to minimize the error on a given task. Generally we consider the problem of minimizing a *loss function* $L : \mathbb{R}^m \to \mathbb{R}$:

$$\min_{\theta} L(\theta),$$

where θ represents the learnable parameters of the neural network, for example all weights $\mathbf{W}$ and biases $\mathbf{b}$ in the case of the MLP (see 2.1.1). The loss function takes as input the neural network parameters and returns a scalar which measures the performance of the model on the task. Common loss functions include Mean Squared Error (MSE) for regression tasks:

$$L = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{y}_i - \mathbf{y}_{\text{true},i})^2,$$

and Cross-Entropy Loss for classification tasks:

$$L = -\frac{1}{n}\sum_{i=1}^{n}\left[\mathbf{y}_{\text{true},i}\log(\mathbf{y}_i) + (1-\mathbf{y}_{\text{true},i})\log(1-\mathbf{y}_i)\right],$$

where n is the number of samples in the dataset, $\mathbf{y}_i$ is the predicted value (or predicted probability for Cross-Entropy) for the i-th sample, and $\mathbf{y}_{\text{true},i}$ is the true value for the i-th sample.

Once the model error has been measured, parameters θ need to be adjusted to minimize it. Backward propagation, commonly known as *backpropagation* (Rumelhart, Hinton, and Williams, 1986; LeCun, Bottou, Bengio, et al., 1998), is a fundamental algorithm used for training neural networks. It allows calculating the gradient of the loss function with respect to parameters, i.e., computing the first-order partial derivatives of the loss function with respect to each weight and bias in the network, denoted as $\frac{\partial L}{\partial \theta}$. Using the gradient computed with backpropagation, the network parameters are adjusted via *gradient descent*. The update rule is:

$$\theta_{t+1} := \theta_t - \alpha\frac{\partial L}{\partial \theta_t},$$

where α is the learning rate: a hyperparameter of the learning algorithm guiding the magnitude of adjustments made to θ_k. This process of computing the model loss and adjusting its

parameters by taking a gradient descent step (as illustrated in Figure 2.2) is repeated iteratively until satisfactory results are obtained or the computational budget is exhausted.

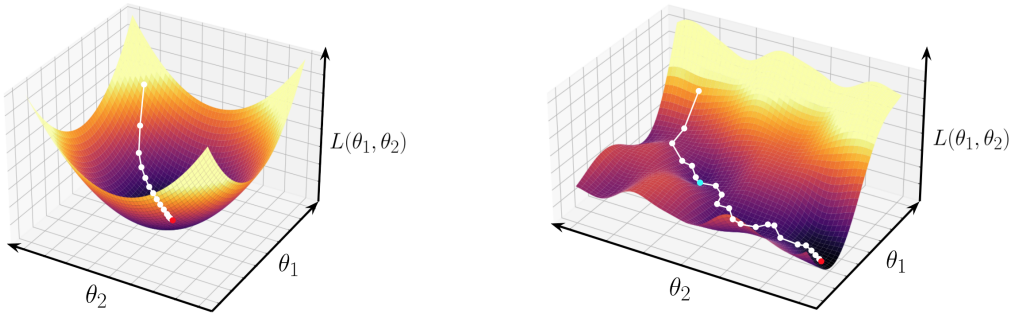


Figure 2.2 – A simple case of gradient descent with only two parameters: θ_1 and θ_2, in the case of a perfectly convex loss function (left); a non-convex loss function (right) with a local minimum (blue dot).

In practice, pure gradient descent using all examples in the dataset to compute the loss is inefficient and unscalable, because processing large datasets containing massive amounts of data requires excessive computational resources. A common variant of gradient descent is Stochastic Gradient Descent (SGD) (Robbins and Monro, 1951; Bottou, 2010), which updates the parameters using only a subset of the training data (a mini-batch) rather than the entire dataset. SGD makes the optimization process computationally tractable, but also prone to escape local minima, potentially leading to better solutions in non-convex optimization problems. The update rule for SGD is:

$$\theta := \theta - \alpha \frac{\partial L_{\text{mini-batch}}}{\partial \theta},$$

where $L_{\text{mini-batch}}$ is the loss computed on a mini-batch of the data. Figure 2.2 illustrates gradient descent in the case of a perfectly convex loss function (left), and a non-convex loss function (right). SGD guarantees convergence to the global minimum in the convex case. However, most neural network loss functions are non-convex, and while SGD does not guarantee finding the global minimum, it often finds a good local minimum due to its stochastic nature.

Remark 2.2 (Local minima and exploration)**.** *In some cases, it is already a hard-exploration problem to escape the local minima of a loss function in supervised learning. Properly tuning the learning rate and using stochastic gradient descent to explore the loss function landscape may help to find the global minimum (see Figure 2.2).*

Several advanced optimization algorithms build on SGD to improve convergence speed and robustness, such as momentum based gradient descent (Polyak, 1964), AdaGrad (Duchi, Hazan, and Singer, 2011), RMSprop (Tieleman and Hinton, 2012) and Adam (Kingma and

Ba, 2014). In this thesis we mainly use the Adam optimizer, which combines the advantages of RMSprop and AdaGrad by computing adaptive learning rates for each parameter from estimates of first and second moments of the gradients. The parameter update rule is defined as follows:

$$\theta_t := \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon},$$

where $\hat{m}_t$ and $\hat{v}_t$ are bias-corrected versions of the first and second moment estimates of the gradients respectively, defined as:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t},$$

and:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t},$$

where g_t is the gradient of the loss function at timestep t, and β_1 and β_2 are exponential decay rates for the first and second moment of the gradient, respectively. Intuitively, m_t is an exponentially moving average of past gradients, which acts like momentum. Similarly, v_t is an exponentially moving average of the squared gradients, which captures the variance (or spread) of the gradients and helps to adapt the learning rate according to how much the gradients are fluctuating.

2.2 Quality-Diversity

In this section, we present the base formalism for quality-diversity (QD) optimization algorithms. We start with an introduction to the general evolutionary framework, and then explore quality-diversity.

2.2.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a class of optimization algorithms inspired by the process of natural selection. They normally operate on a population of potential solutions, and apply the principles of evaluation, selection, and mutation (and/or crossover) to evolve better solutions over time. The general functioning of a classical evolutionary algorithm can be formalized as follows:

1. **Initialization**: Creation of an initial population $\mathcal{P}_0$ containing N random solutions:

$$\mathcal{P}_0 = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}, \quad \mathbf{x}_i \in \mathbb{R}^n.$$

   Each individual (or genotype) $\mathbf{x}_i$ of the population is usually represented as a vector of parameters and constitutes a candidate solution to the problem. Of course, the parameter representations depend on the problem being addressed. For instance, they can range from binary strings that are solutions to a boolean satisfiability problem, to weights and biases of a neural network.

2. **Fitness evaluation**: Evaluation of the fitness $f(\mathbf{x}_i)$ in the environment for each individual in the population. Here, f maps a solution $\mathbf{x}$ to a scalar value, capturing its performance on the given problem:

$$f : \mathbb{R}^n \to \mathbb{R}.$$

3. **Selection**: Selection of individuals that will be modified through mutation and/or crossover. Generally, the probability P of selecting an individual is proportional to its fitness:

$$\mathcal{S} = \{\mathbf{x}_i \mid P(\mathbf{x}_i) \propto f(\mathbf{x}_i)\},$$

   but encouraging diversity by also selecting less effective solutions that present interesting behaviors is often beneficial.

4. **Mutation and crossover**: Modification of individuals in $\mathcal{S}$: 1. Random mutations promote genetic diversity, mutations methods include bit-flip mutation for binary representations and Gaussian mutation for real-valued representations, 2. Crossover pairs individuals (parents) from $\mathcal{S}$ to produce offspring that inherits parents traits.

$$\text{crossover}(\mathbf{x}_i, \mathbf{x}_j) \to \{\mathbf{x}'_i, \mathbf{x}'_j\}$$
$$\text{mutation}(\mathbf{x}_i) \to \mathbf{x}'_i$$

5. **Replacement**: Creation of the new population $\mathcal{P}_{t+1}$ by selecting a combination of existing individuals from the current population and new offspring:

$$\mathcal{P}_{t+1} = \{\mathbf{x}_i \mid \mathbf{x}_i \in \text{offspring} \cup \text{selected current population}\}$$

   This approach ensures both the preservation of good solutions from the current population and the introduction of new genetic material.

This whole process (from step 2 to 5) is repeated until a termination condition is met. For instance, termination conditions include reaching a fitness threshold or a maximum number of iterations. Evolutionary algorithms are considered derivative-free optimization methods,

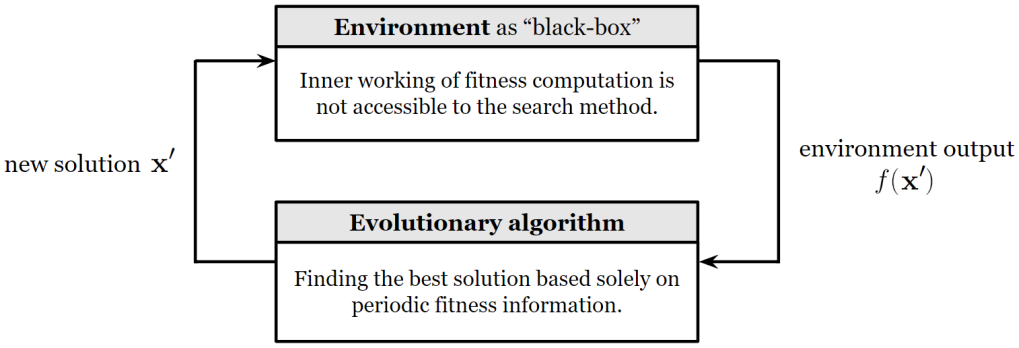


Figure 2.3 – Evolutionary algorithms are derivative-free black-box optimization methods.

meaning they do not involve computing gradients and do not need access to the inner working of the fitness function. Instead, EAs see the environment — the component that allows solutions to be tested against the problem to compute fitness — as a black box, relying solely on the fitness evaluations to guide the search process (as illustrated in Figure 2.3). This particularity makes EAs useful for optimization problems where the underlying structure of the fitness function is complex, non-differentiable or unknown.

2.2.2 Quality-Diversity Optimization

A fascinating aspect of nature lies in its ability to produce a large and diverse collection of organisms that are all high-performing in their niche. By contrast, most evolutionary algorithms focus on finding a single efficient solution to a given problem, a notable difference that cuts short the analogy between evolutionary algorithms and natural evolution (Mouret, 2020). In the real world, although all creatures share the same planet and are part of the same evolutionary process, they are not all in competition with each other. For example, it does not make sense to say that butterflies are more efficient than bears because they do not compete in the same ecological niche. Each of these species has unique characteristics that contribute to their survival and reproduction. Intuitively, Quality-Diversity (QD) algorithms build on classical evolutionary methods and aim to reproduce the mechanisms of natural evolution (Pugh, Soros, and Stanley, 2016; Chatzilygeroudis, Cully, Vassiliades, et al., 2021). While classical optimization methods focus on finding a single high-performing solution, the role of QD methods is to cover the range of possible solution types and to return the best solution for each type, producing a *collection*[1] of solutions rather than a unique one. This process is sometimes referred to as "illumination" in opposition to optimization, as the goal of these algorithms is to reveal (or illuminate) a search space of interest (Mouret and Clune, 2015).

[1] In the QD literature, this collection is often called the archive, the repertoire or even set of solutions. In this work we will use these terms interchangeably.

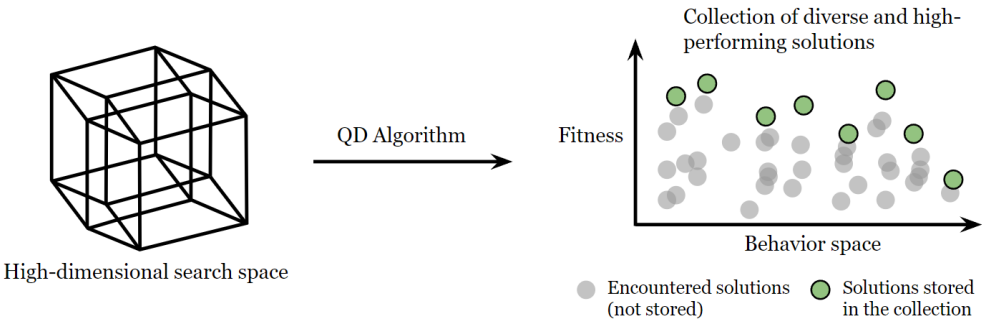


Figure 2.4 – QD algorithms are historically evolutionary algorithms that project a high-dimensional search space (the space of solution genotypes) into a lower-dimensional space capturing the solution behaviors. Here, for the sake of simplicity, the behavior space is unidimensional. Taken from Cully and Demiris (2017).

This search space of interest is a cornerstone of QD approaches. For example, one might be interested in evolving walking robots that differ in their gaits. In this case, a good criterion to differentiate solutions within a population could be the frequency with which a robot puts its legs on the ground, the average height of its center of gravity, or any other relevant criterion that allows for differentiation of solutions in a desired manner. In this example, the goal of a QD algorithm is to evolve walking robots while spanning the space of gaits (the search space of interest), returning the best walking robot for each gait niche. In the QD literature, this search space of interest is often called the *behavior space*, or *behavior descriptor space*, or simply *descriptor space*. In this thesis, we will use these terms interchangeably, but formally, we say that a descriptor (or set of descriptors, usually in the form of a real-value vector) is used to capture the behavior of a solution. Figure 2.4 gives an high-level abstraction of the QD optimization process: a QD algorithm projects a high-dimensional search space (the space of genotypes that encode the solutions)[2] into a lower dimensional space (the behavior space) defined by the solution descriptors. The goal of this projection is to create a collection of solutions that covers the behavior space, and in which the best solution for each part of the behavior space is retained.

Remark 2.3 (Behavior coverage)**.** *Uniform exploration of the genotype space (high-dimensional search space in Figure 2.4) does not guarantee uniform exploration of the behavior space, otherwise generating diversity in behavior space would be trivial, requiring nothing more than random sampling in genotype space. Instead, it is often the case that the most interesting and diverse solutions in the behavior space are contained in low-dimensional manifolds embedded in the genotype space (Rakicevic, Cully, and Kormushev, 2021).*

[2]Up to millions of dimensions in the case where solutions to the problem are deep neural networks, each parameter of the neural network being a search dimension.

More formally, let $\mathbf{x} \in \mathcal{S}$ represent a solution ($\mathcal{S}$ being the solution space), $\mathcal{B}$ the behavior space, $\xi : \mathcal{S} \to \mathcal{B}$ the behavior extraction function that maps a solution to its behavior descriptor $\xi(\mathbf{x})$, and $f : \mathcal{S} \to \mathbb{R}$ the fitness function that maps a solution to its performance $f(\mathbf{x})$. The objective of a QD algorithm is to find a set (or collection) of solutions $\mathcal{X} \subseteq \mathcal{S}$ that maximizes:

$$\max_{\mathcal{X} \subseteq \mathcal{S}} \left\{ \sum_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) \right\} \quad \text{subject to} \quad \mathcal{X} \text{ is diverse}, \tag{2.3}$$

where $\mathcal{X}$ is considered diverse if the set $\{\xi(\mathbf{x}) \mid \mathbf{x} \in \mathcal{X}\}$ of unique behavior descriptors of solutions spans a wide range of values, and where:

$$\sum_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}),$$

is called the QD score, i.e., the sum of fitnesses of all solutions contained in the collection $\mathcal{X}$. As stated in Equation 2.3, this score alone is insufficient to measure the overall performance of a QD algorithm since it can be hacked by finding a single solution with extremely high fitness, which contradicts the diversity objective. A measure of diversity can be formally expressed as:

$$\text{Diversity}(\mathcal{X}) = \left| \{\xi(\mathbf{x}) \mid \mathbf{x} \in \mathcal{X}, \xi(\mathbf{x}) \text{ unique}\} \right|,$$

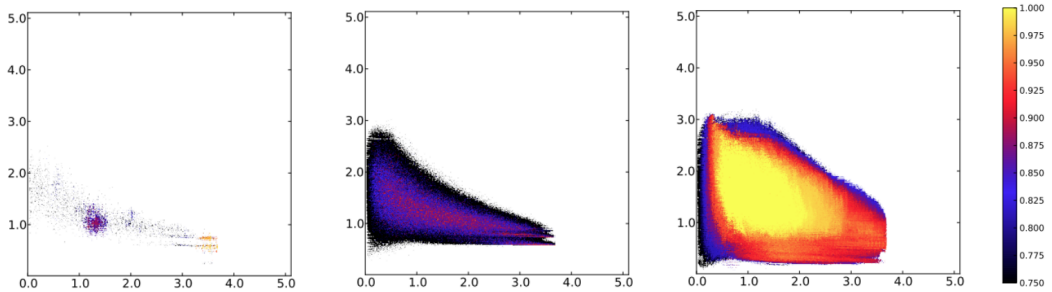
where the set $\{\xi(\mathbf{x}) \mid \mathbf{x} \in \mathcal{X}, \xi(\mathbf{x}) \text{ unique}\}$ represents the unique behavior descriptors of the solutions in $\mathcal{X}$, but such a metric only considers the absolute number of different solutions found by the algorithm, and not the actual *coverage* of the behavior space. In quality-diversity, the coverage refers to the proportion of the behavior space that is effectively covered by the solutions in the collection. To calculate this proportion, it is first necessary to partition the behavior space, i.e., to divide it into discrete regions (or cells). Let $\mathcal{G}$ be a discretiezd grid of the behavior space $\mathcal{B}$ and let $\mathcal{G}_{\text{occupied}} \subseteq \mathcal{B}$ be the set of grid cells occupied by at least one solution in $\mathcal{X}$, coverage can be defined as:

$$\text{Coverage}(\mathcal{X}) = \frac{|\mathcal{G}_{\text{occupied}}|}{|\mathcal{G}|}. \tag{2.4}$$

Thus, the goal of QD algorithms is to ensure that a wide range of behaviors are covered, while high-quality solutions are found for as many different behaviors as possible.

The motivation behind quality-diversity is to address complex problem-solving scenarios where a diverse set of high-quality solutions can provide advantages. For example, in robotics, having multiple strategies for navigating an environment can enhance robustness and adaptability (Cully, Clune, Tarapore, et al., 2015). In design and creativity tasks, diverse solutions can inspire innovation and offer a variety of effective approaches to a problem (Gravina, Khalifa, Liapis, et al., 2019; Fontaine and Nikolaidis, 2021; Sudhakaran, González-Duque, Freiberger,

et al., 2024). Furthermore, aiming for diversity in addition to performance is a convenient way to avoid local optima and explore the solution space more effectively (Colas, Sigaud, and Oudeyer, 2018; Colas, Madhavan, Huizinga, et al., 2020).



(a) Traditional EA (b) Random Sampling (c) MAP-Elites

Figure 2.5 – MAP-Elites produces significantly higher-performing and more diverse solutions than control algorithms for the pattern recognition task described in Clune, Mouret, and Lipson (2013). x and y axes represent the arbitrary space of interest to explore, while color indicates performance. Taken from Mouret and Clune (2015).

Historically, Novelty Search (NS) (Lehman and Stanley, 2011a) is a precursor to Quality-Diversity algorithms that focuses exclusively on exploring novel behaviors. Instead of optimizing for performance, Novelty Search rewards solutions for exhibiting behaviors that are different from those seen before. Formally, Novelty Search aims to maximize the novelty score ρ defined as:

$$\rho(\mathbf{x}) = \frac{1}{k}\sum_{i=1}^{k} \text{distance}(\xi(\mathbf{x}), \xi(\mathbf{x}_i)), \tag{2.5}$$

where $\mathbf{x}_i$ are the k-nearest neighbors of $\mathbf{x}$ in the behavior space. As shown in Figure 1.4, NS beats traditional evolutionary algorithms to solve the hard-exploration maze without ever focusing on the actual objective. Interestingly, NS has been shown to asymptotically act like a uniform random search in the behavior space (Doncieux, Laflaquière, and Coninx, 2019), an important property that allows diversity-oriented algorithms to explore arbitrary spaces of interest that cannot be directly explored by random search in the solution (or genotype) space (see Remark 2.3). In their work, Doncieux, Laflaquière, and Coninx (2019) also introduce the notion of alignment between the behavior space and the task. They show that performing NS on a behavior space that is unaligned with the task at hand does not necessarily result in a uniform exploration of the task space, hence the importance of carefully choosing the behavior space. For example, returning to the maze problem, if one wants to run Novelty Search to find the exit of a maze and defines the behavior space as the final position of a robot in the maze, there is a reasonable chance that the search process ends up exploring the whole task space by seeking diversity in the behavior space. However, if the behavior space is defined as the final

orientation of the robot in the maze (i.e. where is the robot looking at the end of an episode), behavior space is unaligned with the task, and seeking diversity in the final orientations of the robot will not result in a good exploration of the task space.

Following the original introduction of Novelty Search, Novelty Search with Local Competition (NSLC) (Lehman and Stanley, 2011b) can be considered as the first quality-diversity algorithm. NSLC extends Novelty Search by introducing a local competition mechanism, where individuals compete not only based on their novelty but also on their fitness within a local neighborhood in the behavior space. This approach ensures that diversity is encouraged, and that solutions also need to perform well relative to their similar neighbors. As in traditional EAs, NS and NSLC evolve a population of candidate solutions throughout the search process, but unlike traditional EAs, NS and NSLC also maintain an archive that 1. stores all solutions from the evolving population that present enough novelty, 2. is used to compute the novelty score (see Equation 2.5) by comparing new solutions in the population and stored solutions in the archive, and 3. constitutes the final collection of diverse solutions that is returned by the algorithm. In NS-based algorithms, the archive is a different and separate object from the population that only comes into play when calculating the novelty score. It is also continuous, unstructured and based on the Euclidean distance between solution descriptors, meaning that it grows as new novel solutions are found without imposing a predefined structure or partitioning the behavior space.

Algorithm 1 MAP-Elites

Given Max iteration I, Number of initial solutions G, MAP-Elites archive $\mathbb{M}$
$iteration\_number \leftarrow 0$
while $iteration\_number < I$ **do** ▷ Main loop
  if $iteration\_number < G$ **then** ▷ Initialization
    $\mathbf{x}' \leftarrow random\_solution()$
  else
    $\mathbf{x} \leftarrow random\_selection(\mathbb{M})$
    $\mathbf{x}' \leftarrow random\_genetic\_mutation(\mathbf{x})$
  $evaluate(\mathbf{x}')$ ▷ Compute fitness and descriptors
  if fitness of $\mathbf{x}'$ is higher than the corresponding solution in $\mathbb{M}$ **then**
    $insert(\mathbf{x}', \mathbb{M})$
  $iteration\_number \leftarrow iteration\_number + 1$

In this thesis, we mainly build on the Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) (Mouret and Clune, 2015), a pillar algorithm of QD approaches that has been reused and studied in numerous works (Colas, Madhavan, Huizinga, et al., 2020; Flageat and Cully, 2020; Fontaine, Togelius, Nikolaidis, et al., 2020; Nilsson and Cully, 2021). MAP-Elites chronologically follows NSLC but differs in several key aspects. Unlike NS-based algorithms, MAP-Elites defines the archive for storing solutions as a discrete grid (similar to $\mathcal{G}$ in Equation 2.4), dividing the behavior space into discrete cells. Each cell in the grid represents a niche and

stores a unique solution, considered an elite in this part of the behavior space. In contrast to NS, the MAP-Elites archive also serves as the population evolved by the search process, making MAP-Elites a surprisingly simple yet powerful method. The full algorithm is described in Algorithm 1 and can be summarized as follows: 1. MAP-Elites starts with an empty archive and a random set of initial solutions that are evaluated and placed into the archive according to a simple insertion criteria: if the cell corresponding to a solution's behavior descriptor is unoccupied, the solution is inserted into that cell. If there is already a solution in this cell, the new solution will only replace it if it has superior fitness. 2. At every iteration, a set of existing solutions are randomly selected from the repertoire and mutated to generate new solutions. 3. These new solutions are then evaluated and inserted into the repertoire using the same insertion criteria as before. This cycle is repeated until either convergence is reached or a predetermined number of iterations is completed. Figure 2.5 illustrates the coverage capacity of MAP-Elites in comparison to a traditional evolutionary algorithm and random sampling of the solution space. The selection operator of MAP-Elites (2) plays a crucial role in its capacity to cover the behavior space, Mouret and Clune (2015) show that it helps to promote serendipity in the search process (i.e. the elite in a given cell often comes from solutions present in distant regions of the behavior space that has been mutated), an important property that is lost when the selection is biased towards the solutions that present higher fitnesses.

As shown by Cully and Demiris (2017), there is a wide variety of possible QD algorithms that can be drawn between the main families of Novelty Search and MAP-Elites, which differ considering the archive type, selection operator, and considered value for selection. But importantly, although quality-diversity is historically based on evolutionary methods, it is not theoretically restrained to them and can qualify any form of optimization that aims to produce a diverse collection of solutions (Arulkumaran, Cully, and Togelius, 2019), such as information-theory-augmented RL methods (Eysenbach, Gupta, Ibarz, et al., 2018; Sharma, Gu, Levine, et al., 2019; Kumar, Kumar, Levine, et al., 2020; Chalumeau, Boige, Lim, et al., 2022).

2.3 Reinforcement Learning

In what follows, we introduce the formalism of reinforcement learning (RL), largely inspired by Sutton and Barto (2018), whose work is foundational in this field. We start by introducing the class of problems considered in RL, and then present key RL concepts and algorithms that we use in this thesis.

2.3.1 Base Formalism

Markov decision processes (MDPs) (Howard, 1960) provide the mathematical framework for formulating RL problems. An MDP is generally defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where:

1. **State space** (S): $\mathcal{S}$ is the space of all possible states the environment can be in. A state $s_t \in \mathcal{S}$ contains all information about the environment at timestep t. $\mathcal{S}$ can be either discrete or continuous depending on the problem considered. For example, in a board game like chess, the state space is discrete as it consists of a finite number of board configurations. In contrast, in robotic control (which is the main type of environment used in this thesis), the state space is continuous as it includes parameters like positions and velocities, which can take any real value.

2. **Action space** ($\mathcal{A}$): $\mathcal{A}$ is the space containing all possible actions the agent can take. At each timestep, the agent usually choose an action $a \in \mathcal{A}$ that affects the future environment state. For example, in a robot navigation task, actions might include moving north, south, east, or west. As with the state space, $\mathcal{A}$ can be continuous or discrete depending on the task at hand.

3. **Transition model** ($\mathcal{P}$): $\mathcal{P}$ is the transition model that defines the probability distribution over the next state given the current state and action. $\mathcal{P}(s' \mid s, a)$ is the probability of transitioning to state s' from state s when a is the action taken by the agent. For example, in a stochastic world, the action taken by the agent may not always give the desired outcome due to environment uncertainty or noise.

4. **Reward function** ($\mathcal{R}$): $\mathcal{R}$ is the reward function that returns a reward $\mathcal{R}(s, a, s')$ for transitioning from state s to state s' via action a. The reward is a feedback given to agent providing information about the performance of its actions.

A fundamental characteristic of MDPs is that the transition function $\mathcal{P}$ is Markovian, meaning future states of the process only depend on the current state s_t and action a_t, and not on the sequence of events that occurred previously:

$$\mathcal{P}(s_{t+1} \mid s_t, a_t) = \mathcal{P}(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0),$$

for all states s_t, s_{t+1} and actions a_t. The Markov property in important for reinforcement learning as it reduces the complexity of modeling transitions and simplifies practical implementation, particularly in large —and continuous— state and action spaces. Instead of considering the entire trajectory of states and actions, only the current state and action are needed. For instance, a state s_t containing the position and velocity of a robot is more compact and easier to deal with than a multiple states $s_t, s_{t-1}, \ldots, s_0$ containing only robot positions, from which an agent would need to infer its current speed. The Markov property is also involved in convergence and stability guarantees of some RL algorithms such as SARSA and Q-learning and SARSA (Watkins and Dayan, 1992; Sutton and Barto, 2018). In this thesis, we consider finite-horizon MDPs, where the agent plays finite episodes in the environment before being reset in a starting state. Figure 2.6 illustrates the basic operation of reinforcement learning in MDPs: At timestep

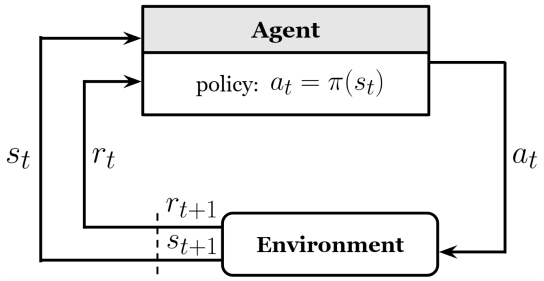


Figure 2.6 – The RL agent-environment interaction loop in a Markov decision process, taken from Sutton and Barto (2018). The environment provides states and rewards, and the agent policy chooses actions.

t, the agent receives the state s_t and reward r_t from the environment, generates an action a_t, and sends it back to the environment. The environment then computes the next state s_{t+1} and reward r_{t+1} using a_t.

In reinforcement learning, a *trajectory* τ (or *episode*) is a finite sequence of states, actions and rewards that the agent encounters while interacting with the environment from the initial state s_0 to a terminal state s_T. Formally, it can be represented as:

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_T). \tag{2.6}$$

The *return* R_t is the total accumulated reward on a trajectory τ from timestep t until the end of the episode T. Denoting r_t the reward as timestep t, the return can be formulated as follows:

$$R_t(\tau) = \sum_{k=t}^{T} r_k.$$

In practice, we generally use a discounted version of the return to prioritize immediate rewards over those received further in the future:

$$R_t(\tau) = \sum_{k=t}^{T} \gamma^{k-t} r_k,$$

where γ is the discount factor $0 \leq \gamma < 1$ reducing the value of future rewards. The intuition behind using a discounted return is to account for the uncertainty of future rewards, ensure computational tractability (mostly in the case of infinite-horizon MDPs), encourage immediate feedback and reflect the time preference inherent in many real-world scenarios.

A *policy* π is a strategy used by the agent to decide the next action a given the current state s. It can be deterministic:

$$\pi(s) = a,$$

or stochastic, where it defines a probability distribution over actions:

$$\pi(a \mid s) = P(A_t = a \mid S_t = s).$$

The policy plays a central role in RL as it defines the agent behavior. It can be parameterized by θ, where π_θ denotes the policy with parameters θ. In modern RL —as well as in this thesis—, policies are often implemented as neural networks, allowing to handle high-dimensional state and actions spaces effectively. For discrete action spaces, the neural network typically outputs a probability distribution over the possible actions:

$$\pi_\theta(a \mid s) = \mathrm{softmax}(f_\theta(s)),$$

where $f_\theta(s)$ is the output of the model before applying the softmax. For continuous action spaces, the neural network outputs the parameters of a probability distribution, such as the mean and standard deviation for a Gaussian distribution, from which actions are sampled:

$$a \sim \mathcal{N}(\mu_\theta(s), \sigma_\theta(s)),$$

where $\mu_\theta(s)$ and $\sigma_\theta(s)$ are the mean and standard deviation of the Gaussian distribution output by the neural network.

The *goal of reinforcement learning* is to find an optimal policy π^* that maximizes the expected return J:

$$\pi^* = \arg\max_{\pi \in \Pi} J(\pi),$$

with Π denoting the space of policies. The expected return of π is defined as:

$$J(\pi) = \mathop{\mathbb{E}}_{\tau \sim \pi}[R_0(\tau)], \tag{2.7}$$

where $\tau \sim \pi$ indicates that trajectories τ are generated by following policy π. In our case, considering policies as deep neural networks parameterized by θ, this involves iteratively updating the parameters θ to maximize the expected return. The optimization problem can be formulated as:

$$\theta^* = \arg\max_{\theta} \mathop{\mathbb{E}}_{\tau \sim \pi_\theta}[R_0(\tau)].$$

2.3.2 Value Functions

In RL, value functions estimate the expected return of states (or state-action pairs) under a policy π. The state value function $V^\pi(s)$ is the expected return starting from state s and

following policy π afterwards. Put simply, it captures the value of being in the state s under the policy π:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau \sim \pi} \left[R_t(\tau) \mid s_t = s \right]. \tag{2.8}$$

Similarly, the state-action value function $Q^{\pi}(s, a)$ is the expected return starting from state s, taking action a and then following π:

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{\tau \sim \pi} \left[R_t(\tau) \mid s_t = s, a_t = a \right].$$

The *Bellman equations* (Bellman, 1966) provide a recursive decomposition of value functions, which is fundamental in many RL problems. For instance, the Bellman expectation equation for $V^{\pi}(s)$ derives from Equation 2.8 and expresses the value of s in terms of the value of subsequent states:

$$\begin{aligned}
V^{\pi}(s) &= \mathop{\mathbb{E}}_{\tau \sim \pi} \left[R_t(\tau) \mid s_t = s \right] \\
&= \mathop{\mathbb{E}}_{\tau \sim \pi} \left[r_{t+1} + \gamma R_{t+1}(\tau) \mid s_t = s \right] \\
&= \sum_{a \in A} \pi(a \mid s) \sum_{s' \in S} \mathcal{P}(s' \mid s, a) \left[r_{t+1} + \gamma \mathop{\mathbb{E}}_{\tau \sim \pi} \left[R_{t+1}(\tau) \mid s_{t+1} = s' \right] \right] \\
&= \sum_{a \in A} \pi(a \mid s) \sum_{s' \in S} \mathcal{P}(s' \mid s, a) \left[r_{t+1} + \gamma V^{\pi}(s') \right].
\end{aligned} \tag{2.9}$$

In simple words, Equation 2.9 expresses the value function $V^{\pi}(s)$ as the probability of choosing action a given s, the probability of ending in a next state s' after executing action a, the immediate reward r_{t+1} received for doing action a in state s, and the discounted value function of the next state $V^{\pi}(s')$, summed over all $a \in \mathcal{A}$ and all $s' \in \mathcal{S}$. In a related way, the *Bellman expectation equation* provides a recursive definition for the state-action value function $Q^{\pi}(s, a)$:

$$Q^{\pi}(s, a) = \sum_{s' \in S} \mathcal{P}(s' \mid s, a) \left[r_{t+1} + \gamma \sum_{a' \in A} \pi(a' \mid s') Q^{\pi}(s', a') \right].$$

Finally, the *Bellman optimality equations* describe the value functions for the optimal policy π^*. These equations provide a way to find the optimal policy by expressing the value functions in terms of the best possible actions at each state. The Bellman optimality equation for $V^*(s)$ represents the maximum value achievable from state s following the optimal policy:

$$V^*(s) = \max_a \sum_{s' \in S} \mathcal{P}(s' \mid s, a)[r_{t+1} + \gamma V^*(s')],$$

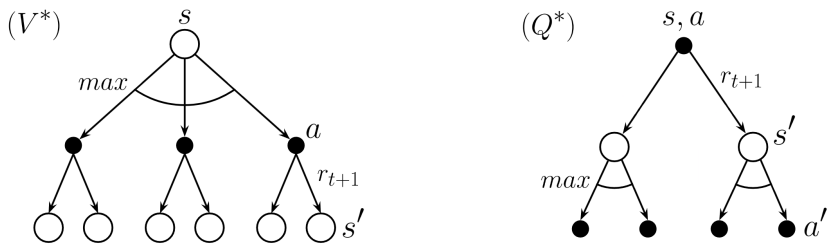


Figure 2.7 – Backup diagrams for the Bellman optimality equations for $V^*(s)$ and $Q^*(s, a)$. Backup diagrams provide intuitive interpretations of Bellman equations, where states are represented as large empty circles and state-actions pairs as smaller black dots. Arrows denote the possible scenarios following either a state or a state-action pair.

and the Bellman optimality equation for $Q^*(s, a)$ represents the maximum value achievable by taking action a in state s and then following π^*:

$$Q^*(s, a) = \sum_{s' \in S} P(s' \mid s, a) \left[r_{t+1} + \gamma \max_{a'} Q^*(s', a') \right].$$

Figure 2.7 provides an illustration for Bellman optimality equations.

The preceding sections explored foundational frameworks of Markov Decision Processes (MDPs) and value functions as tools for understanding environment dynamics and potential rewards. The next sections transition to core methodologies that enable an agent to effectively learn optimal policies and aim to answer the following question: **How to learn with reinforcement learning?**. These methods build upon the theoretical constructs previously discussed and introduce approaches for refining and optimizing the agent decision-making process over time.

2.3.3 Temporal Difference Learning

Temporal difference (TD) (Sutton, 1988) Learning is a central technique in reinforcement learning aiming at computing increasingly accurate estimations of the value functions. TD-based algorithms learn optimal policies by updating estimates of the state values (resp. state-action values) based on the differences between estimated values at successive states. This method bridges the gap between Monte Carlo methods (Metropolis and Ulam, 1949; Sutton, 1988), which require the completion of entire episodes to make updates, and dynamic programming (Bellman, 1952, 1966), which assumes a full knowledge of the environment's dynamics. TD learning performs incremental updates after each step of an episode using:

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]. \tag{2.10}$$

This formula allows the agent to update its knowledge of the state value based not on the actual return over the full episode, as in Monte Carlo methods, but on the estimated value of the next state $V(s')$. Intuitively, $r + \gamma V(s')$ in Equation 2.10 represents the *target* (or in some way, the label), which is a little closer to the true value of state s than $V(s)$, since it incorporates the true reward r for the current timestep and relies on estimate $V(s')$ only for the rest of the trajectory. Analogously to Equation 2.10, we can derive the TD update for state-action values as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]. \tag{2.11}$$

By continually adapting the estimates of $V(s)$ (resp. $Q(s,a)$), TD Learning enables the creation of policies that are incrementally better. Each update provides a more accurate reflection of the true value of states and actions, which in turn guides the agent to make optimal choices.

Still, even with a perfect knowledge of $V(s)$, the state value function only contains information about the expected return from each state under an optimal policy. Ideally, a policy aiming at maximizing the return chooses actions that are expected to lead to states with high values. However, $V(s)$ does not specify what these optimal actions are and, more importantly, the transition function $\mathcal{P}$ giving the probability $\mathcal{P}(s' \mid s,a)$ of transitioning to state s' from s when a is the undertaken action is often unknown to the agent, i.e., the agent cannot know which action leads to the desired next state[3]. Generally, in model-free RL, learning $Q(s,a)$ is preferred as it directly informs action selection, bypassing the need for knowledge about the transition dynamics $\mathcal{P}$. Q-learning and SARSA specifically utilize $Q(s,a)$ to adapt and refine policies through direct interaction with the environment, enabling learning without a model of the environment dynamics. Algorithm 2 provides an illustration of the TD-based SARSA algorithm, where an ε-greedy policy is a policy that takes the best estimated action with probability $1-\varepsilon$ to force some degree of exploration.

Algorithm 2 SARSA Algorithm

Given Total episodes E, Learning rate α, Discount factor γ, Exploration policy (e.g., ε-greedy)
Initialize $Q(s,a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
for $episode = 1$ to E **do** ▷ Loop for each episode
    Initialize state s
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    while state s is not terminal **do** ▷ Loop for each step of the episode
        Take action a, observe reward r and next state s'
        Choose a' from s' using policy derived from Q (e.g., ε-greedy)
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$ ▷ SARSA update, Equation 2.11
        $s \leftarrow s'$ ▷ Move to next state
        $a \leftarrow a'$ ▷ Move to next action

[3] In this thesis, we focus on model-free RL. In model-based RL, where the model of the environment is given or learned (i.e. the agent knows the transition probabilities $\mathcal{P}(s' \mid s,a)$), knowing $V(s)$ is sufficient to learn policies.

2.3.4 Policy Gradient Methods

Policy gradient methods are a class of algorithms in reinforcement learning where the policy is directly optimized through gradient ascent on the expected return. Unlike TD learning that iteratively updates value estimates to inform policy decisions indirectly, policy gradient methods deal with the policy function itself, thus offering a more straightforward path to policy improvement. The policy gradient theorem provides a method to calculate how changes to the parameters θ of a policy π_θ affect the expected return.

To derive the policy gradient, we first need to express the expected return $J(\pi_\theta)$ (see Equation 2.7) in terms of trajectory probabilities. Using the definition of a trajectory from Equation 2.6, the probability of a trajectory τ under the policy π_θ is given by:

$$P(\tau \mid \theta) = \mu(s_0) \prod_{t=0}^{T} \pi_\theta(a_t \mid s_t) \mathcal{P}(s_{t+1} \mid s_t, a_t), \tag{2.12}$$

where μ is the initial state distribution, representing the probability of starting in a particular state $s0$ when an episode begins. The expected return $J(\pi_\theta)$ can then be expressed as:

$$J(\pi_\theta) = \sum_{\tau} P(\tau \mid \theta) R(\tau).$$

To derive the gradient of $J(\pi_\theta)$, we use the log-derivative trick, which states that:

$$\nabla_\theta P(\tau \mid \theta) = P(\tau \mid \theta) \nabla_\theta \log P(\tau \mid \theta).$$

Thus, we can write the gradient of $J(\pi_\theta)$ as:

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \nabla_\theta \sum_{\tau} P(\tau \mid \theta) R(\tau) \\ &= \sum_{\tau} \nabla_\theta P(\tau \mid \theta) R(\tau) \\ &= \sum_{\tau} P(\tau \mid \theta) \nabla_\theta \log P(\tau \mid \theta) R(\tau). \end{aligned} \tag{2.13}$$

Using the definition of a trajectory probability given in Equation 2.12 and the properties of the logarithm, $\log P(\tau \mid \theta)$ can be written as:

$$\begin{aligned} \log P(\tau \mid \theta) &= \log \mu(s_0) + \log \left(\prod_{t=0}^{T} \pi_\theta(a_t \mid s_t) \mathcal{P}(s_{t+1} \mid s_t, a_t) \right) \\ &= \log \mu(s_0) + \sum_{t=0}^{T} \left(\log \pi_\theta(a_t \mid s_t) + \log \mathcal{P}(s_{t+1} \mid s_t, a_t) \right). \end{aligned}$$

Taking the gradient of $\log P(\tau \mid \theta)$ with respect to θ gives:

$$\begin{aligned}\nabla_\theta \log P(\tau \mid \theta) &= \nabla_\theta(\log \mu(s_0)) + \sum_{t=0}^{T} (\nabla_\theta \log \pi_\theta(a_t \mid s_t) + \nabla_\theta \log \mathcal{P}(s_{t+1} \mid s_t, a_t)) \\ &= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t), \end{aligned} \tag{2.14}$$

since $\mu(s_0)$ and $\mathcal{P}(s_{t+1} \mid s_t, a_t)$ are independent of θ (their gradients are zero). Combining the results from Equation 2.13 and Equation 2.14, the policy gradient becomes:

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \sum_\tau P(\tau \mid \theta) \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right] R(\tau) \\ &= \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau) \right]. \end{aligned} \tag{2.15}$$

This formulation shows that the gradient of the expected return with respect to the policy parameters can be expressed as an expectation over trajectories of the sum of the gradients of the log probabilities of the actions, scaled by the total return of the trajectory. Intuitively, Equation 2.15 expresses that the way to achieve a higher return is by privileging actions that are part of high-return trajectories.

Note that the policy gradient is formulated as an expectation, meaning that, in practice, we can estimate it with a sample mean:

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau),$$

where $\mathcal{D}$ is a dataset of trajectories $\mathcal{D} = \{\tau_i\}_{i=1,\dots,N}$ produced by letting policy π_θ act in the environment. Algorithm 3 illustrates one of the simplest usage of policy gradient in an RL algorithm to iteratively learn a policy: the REINFORCE algorithm (Williams, 1992).

Algorithm 3 REINFORCE Algorithm

Given Total episodes E, Learning rate α, Discount factor γ
Initialize policy parameters θ randomly
for $episode = 1$ to E **do** ▷ Loop for each episode
    Generate an episode $\tau = (s_0, a_0, r_1, s_1, \dots, s_T)$ under π_θ
    for $t = 0$ to $T - 1$ **do** ▷ Loop for each step of the episode
        $R_t \leftarrow \sum_{k=t}^{T} \gamma^{k-t} r_k$ ▷ Calculate return from timestep t
        $\nabla_\theta J(\pi_\theta) \leftarrow \nabla_\theta \log \pi_\theta(a_t|s_t) R_t$ ▷ Gradient of log-prob
        $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$ ▷ Update policy parameters

In policy gradient methods, a significant challenge is the high variance of gradient estimates, which can lead to unstable learning. The return $R(\tau)$ is the cumulative reward over an entire trajectory. Using it directly in the gradient computation can lead to high variance because it aggregates the effects of all actions taken throughout the trajectory. Even the use of the return from timestep t (as in Algorithm 3) still involves high variance because it depends on the particular sequence of events that follow t. This makes the gradient estimate noisy and can slow down learning. One effective technique to reduce this variance is to use the state-action value function $Q^\pi(s, a)$, which captures the *expected* value over all possible future trajectories, instead of the return in the gradient computation. By using $Q^\pi(s, a)$, the policy gradient theorem can be written as:

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) Q^\pi(s, a)\right].$$

The Q-function usually provides an estimate of the expected return starting from a specific state s after taking action a. This more localized and robust feedback reduces the noise in the gradient estimate and focuses on the immediate impact of individual actions.

2.3.5 Actor-Critic Methods

Actor-critic methods are a class of RL algorithms combining the policy gradient approach with value function approximation. Historically, these methods were developed as an extension of the policy gradient approach, with foundational ideas introduced by Barto, Sutton, and Anderson (1983). Actor-critic methods simultaneously optimize both the policy (actor), which is responsible for updating the policy in the direction suggested by the critic, and the value function (critic), which evaluates the actions taken by the actor based on the computed value functions.

Generally, in the actor-critic framework, the *actor* selects actions according to a policy π_θ and updates its parameters using feedback from the critic following the policy gradient:

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) A_t\right],$$

where $A_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ is the advantage at timestep t. Intuitively, the advantage captures the relative benefit (or potentially the relative disadvantage) of the chosen action over the baseline value of the state. On the other side, the *critic* estimates the value function $V(s)$ and updates its parameters w to minimize prediction error as follows:

$$\Delta w = \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))\nabla_w V_w(s_t).$$

Note that this formulation is similar to TD learning of the value function in Equation 2.10.

In this thesis, we mainly use the TD3 algorithm (Fujimoto, Van Hoof, and Meger, 2018), a state-of-the-art actor-critic algorithm introduced to tackle the inherent overestimation of action values observed in standard DDPG (Lillicrap, Hunt, Pritzel, et al., 2015). TD3, or Twin Delayed Deep Deterministic Policy Gradient, modifies the standard actor-critic architecture as follows: First, it employs two Q-value estimators —or critics—, Q_{w1} and Q_{w2}. For each state-action pair (s, a) both critics output an estimate for the state-action value, and the smaller of the two is used to determine the target value y as:

$$y = r + \gamma \min_{i=1,2} Q_{w_i}(s', \pi_\theta(s') + \varepsilon), \tag{2.16}$$

where ε is noise added to the action $a = \pi_\theta(s')$ to ensure exploration in the action space. Second, TD3 update the policy parameters θ less frequently than those of the critic networks (typically twice less frequently). This delay helps to minimize the risks associated with the propagation of estimations errors from the critics value function estimate to the policy. Third and last, TD3 encourages policy smoothing by adding noise to the target policy in the calculation of target state-action values (see ε in Equation 2.16). This noise forces exploration and enhances training stability, ensuring that the policy is tested against a range of actions around what it believes to be the best action. This testing confirms whether the surrounding actions are also good, or if the policy is overfitting to narrow peaks in value estimates.

In TD3, each critic is updated by minimizing the mean squared error between the predicted state-action value $Q(s, a)$ and the target y from Equation 2.16 as:

$$\nabla_{w_i} \frac{1}{N} \sum_{j=1}^{N} (y_j - Q_{w_i}(s_j, a_j))^2.$$

The actor updates its policy using the policy gradient:

$$\nabla_\theta \frac{1}{N} \sum_{j=1}^{N} Q_w(s_j, \mu_\theta(s_j)).$$

2.4 Software

Software implementation of supervised learning methods (Section 2.1), evolutionary algorithms (Section 2.2), and reinforcement learning (Section 2.3) is almost as important as the theoretical background that justifies them. For instance, implementing the methods that are later presented in this thesis (see Chapters 3 and 4) would have been almost impossible — or would have required much more time or better experimenters — in the pre-2015 era where

no machine learning framework existed[4] to enable automatic differentiation. The software underlying experiments in Chapter 3, which introduces an hybrid QD/RL method, was mainly handmade for the sole purpose of these experiments using PyTorch (Paszke, Gross, Chintala, et al., 2017) and MPI (Message Passing Interface Forum, 2021), a tool for high-performance parallel computing.

However, Chapter 4 builds exclusively on the QDax[5] research framework(Chalumeau, Lim, Boige, et al., 2024), a tool to accelerate QD (and also RL) algorithms through hardware accelerators and massive parallelization. QDax is fully open-source and based on Jax (Bradbury, Frostig, Hawkins, et al., 2018), a high-performance machine learning library that combines the familiarity of NumPy (Harris, Millman, Walt, et al., 2020) with just-in-time compilation, enabling automatic differentiation and GPU/TPU support, making it extremely efficient for large-scale numerical computations. Figure 2.8 illustrates the global functioning of QDax, where simple atomic building blocks are combined to form custom algorithms.

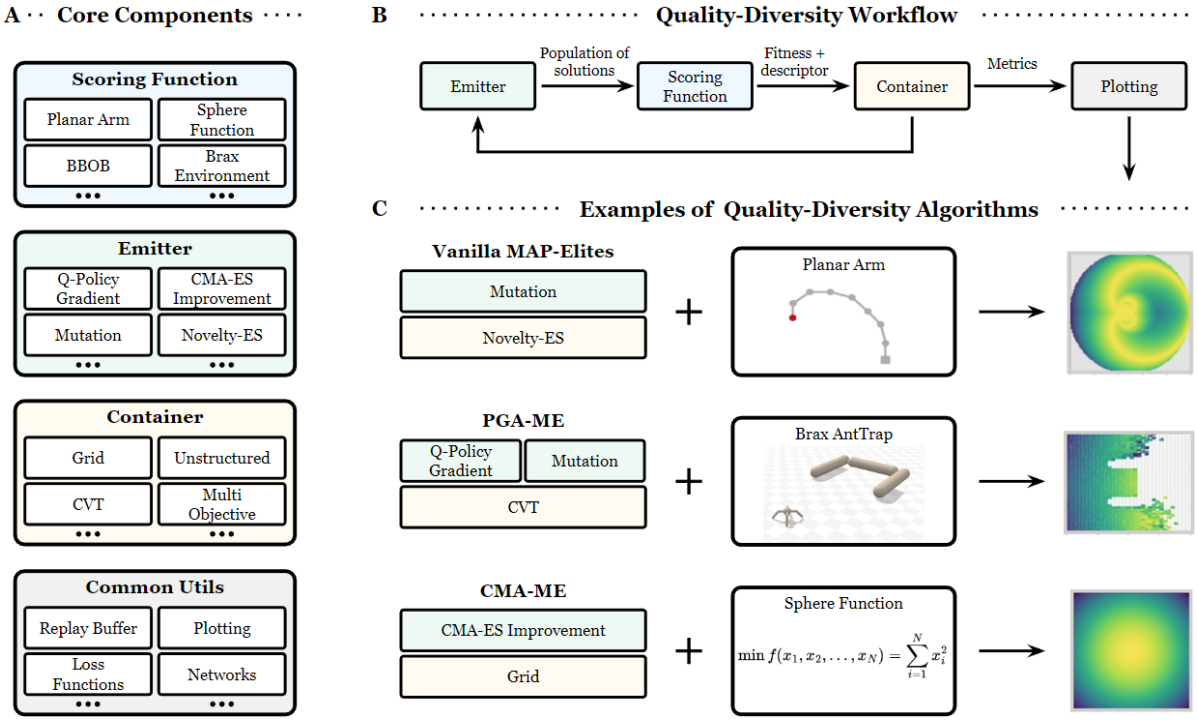


Figure 2.8 – A) Core components, used as building blocks to create optimization experiments. B) High-level software architecture of QDax. C) Various examples of QD algorithms used for a variety of tasks and problem settings available in QDax. Taken from Chalumeau, Lim, Boige, et al. (2024).

[4]TensorFlow (Martín Abadi, Ashish Agarwal, Paul Barham, et al., 2015), arguably the first deep learning framework, was released in 2015 and its first version is notoriously known for its difficulty and steep learning curve.

[5]QDax is the result of a collaboration between InstaDeep and the Imperial College of London, and is the main work of Felix Chalumeau, Raphaël Boige and Bryan Lim. In the context of this thesis, some contributions have been made to QDax.

In QDax, the *container* defines the way the population is stored, the *emitter* implements techniques to evolve solutions from the population (such as mutation-based updates, policy-gradient-based updates, etc.), and the *scoring function*, analogous to the *environment* allows the user to define the optimization problem. QDax includes numerous state-of-the-art QD and RL baselines, whose implementations have been validated through replications of results from the literature.

Chapter 3

Diversity Policy Gradient for Sample Efficient Quality-Diversity Optimization

La pensée ne doit jamais se soumettre,
ni à un dogme, ni à un intérêt, ni a une idée préconçue,
ni à quoi que ce soit, si ce n'est aux faits eux-mêmes;
parce que, pour elle, se soumettre, ce serait cesser d'être.

Henri Poincaré.

This chapter introduces a new algorithm, called QD-PG, which combines the strength of policy gradient algorithms and quality-diversity methods to produce a collection of diverse and high-performing neural policies in continuous control environments. The main contribution is the introduction of a Diversity Policy Gradient (D-PG) exploiting information at the time-step level to drive policies towards more diversity in a sample-efficient manner.

Contents

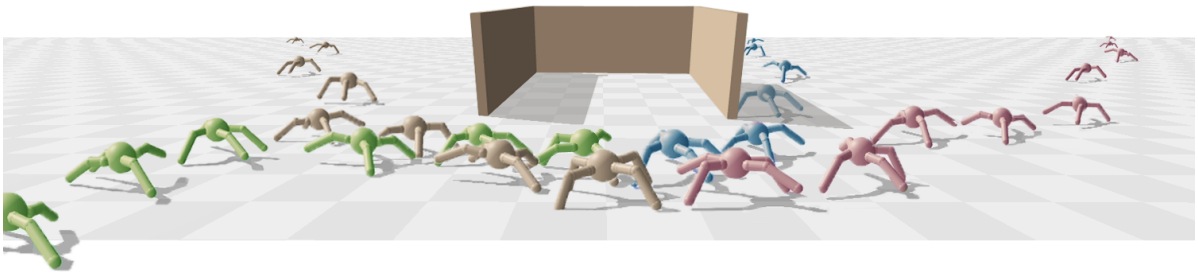


Figure 3.1 – An agent robot is rewarded for running forward as fast as possible. Following the reward signal without further exploration leads into a trap corresponding to a poor local minimum. Our method, QD-PG, produces a collection of solutions that are diverse and high-performing, allowing deeper exploration necessary to solve hard-exploration problems.

3.1 Introduction

The principal advantage of QD approaches resides in their intrinsic capacity to deliver a large and diverse set of working alternatives when a single solution fails (Cully, Clune, Tarapore, et al., 2015). By producing a collection of solutions instead of a unique one, QD algorithms allow to obtain different ways to solve a single problem (see Figure 3.1), leading to greater robustness, which can help to reduce the reality gap when applied to robotics (Koos, Mouret, and Doncieux, 2012). Diversity seeking is the core component that allows QD algorithms to generate large collections of diverse solutions. By encouraging the emergence of novel behaviors in the population without focusing on performance alone, diversity seeking algorithms explore regions of the behavior descriptor space that are unreachable in practice for conventional algorithms (Doncieux, Laflaquière, and Coninx, 2019). Another benefit of QD is its ability to solve hard exploration problems where the reward signal is sparse or deceptive, and on which standard optimization techniques are ineffective (Colas, Madhavan, Huizinga, et al., 2020). This ability can be interpreted as a direct consequence of the structured search for diversity in the behavior descriptor space. As mentioned in Section 2.2.2, quality-diversity algorithms build on black-box optimization methods such as evolutionary algorithms to evolve a population of solutions (Cully and Demiris, 2017). Historically, they rely on random mutations to explore small search spaces but struggle when facing higher-dimensional problems. As a result, they often scale poorly to problems where neural networks with many parameters provide state-of-the-art results (Colas, Madhavan, Huizinga, et al., 2020).

Training large and efficient policies that work with continuous actions has been a long-standing goal in Artificial Intelligence and in particular in robotics. Deep reinforcement learning (RL), and especially Policy Gradient (PG) methods have proven efficient at training such large

policies (Lillicrap, Hunt, Pritzel, et al., 2015; Schulman, Wolski, Dhariwal, et al., 2017; Fujimoto, Van Hoof, and Meger, 2018; Haarnoja, Zhou, Hartikainen, et al., 2018). One of the keys to this success lies in the fact that PG methods exploit the structure of the objective function when the problem can be formalized as an MDP, leading to substantial gains in sample efficiency. Moreover, they also exploit the analytical structure of the policy when known, which allows the sample complexity of these methods to be independent of the parameter space dimensionality (Vemula, Sun, and Bagnell, 2019). In real-world applications, these gains turn out to be critical when interacting with the environment is expensive.

Although exploration is very important to reach optimal policies, PG methods usually rely on simple exploration mechanisms, like adding Gaussian noise (Fujimoto, Van Hoof, and Meger, 2018) or maximizing entropy (Haarnoja, Zhou, Hartikainen, et al., 2018) to explore the action space, which happens to be insufficient in hard exploration tasks (for instance, the maze exploration task depicted in Figure 1.4) where the reward signal is sparse or deceptive (Colas, Sigaud, and Oudeyer, 2018; Nasiriany, Pong, Lin, et al., 2019). Successful attempts have been made to combine evolutionary methods and reinforcement learning (Khadka and Tumer, 2018; Pourchot and Sigaud, 2018; Khadka, Majumdar, Miret, et al., 2019) to improve exploration. However, all these techniques only focus on building high-performing solutions and do not explicitly encourage diversity within the population. In this regard, they fail when confronted with hard exploration problems. More recently, Policy Gradient Assisted MAP-Elites (PGA-ME) (Nilsson and Cully, 2021) was proposed to bring reinforcement learning in MAP-Elites (Mouret and Clune, 2015) to train neural networks to solve locomotion tasks with diverse behaviors. PGA-ME makes use of policy gradient to optimize for performance but still relies on divergent genetic search for exploration, making it struggle on hard exploration tasks.

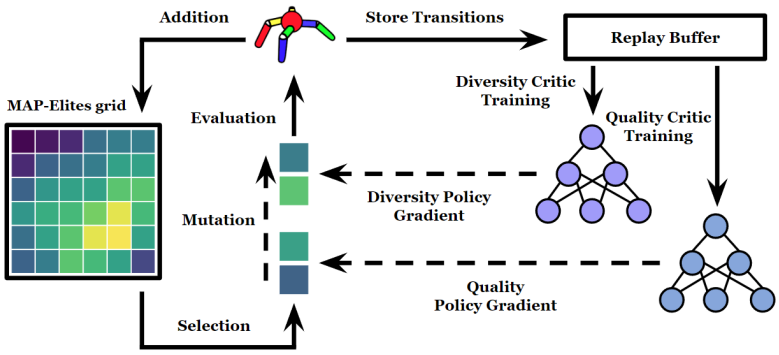


Figure 3.2 – QD-PG builds on the MAP-Elites framework and leverages reinforcement learning to derive policy gradient based mutations.

Contributions. In this work, we introduce the idea of a *diversity policy gradient* (D-PG) that drives solutions towards more diversity. We show that the D-PG can be used in combination with the standard policy gradient, dubbed *quality policy gradient* (Q-PG), to produce high-

performing and diverse solutions. Our algorithm, called QD-PG, replaces random diversity search by D-PG, builds on the general framework of MAP-Elites (ME) and PGA-ME, and demonstrates remarkable sample efficiency brought by off-policy PG methods. We compare QD-PG to state-of-the-art policy gradient methods algorithms (including SAC, TD3, RND, AGAC, DIAYN and PGA-ME), and to several evolutionary methods known as evolution strategies (ESs) augmented with a diversity objective (namely the NS-ES family (Conti, Madhavan, Such, et al., 2018) and the ME-ES algorithm (Colas, Madhavan, Huizinga, et al., 2020)) on a set of hard-exploration continuous control tasks with deceptive reward signal. We show that QD-PG generates collections of robust and high-performing solutions in hard exploration problems while standard policy gradient algorithms struggle to produce a single one. QD-PG is several orders of magnitude more sample efficient than its traditional evolutionary competitors and outperforms PGA-ME on all benchmarks.

3.2 Background

In this section we briefly introduce the theoretical framework for the problem at hand, and motivate our choice for the three methods on which we are building QD-PG, namely MAP-Elites (Mouret and Clune, 2015), TD3 (Fujimoto, Van Hoof, and Meger, 2018), and PGA-MAP-Elites (Nilsson and Cully, 2021). For a more detailed description of MAP-Elites and TD3, please refer to Section 2.2.2 and Section 2.3.5, respectively.

3.2.1 Problem Statement

As presented in Section 2.3.1, we consider Markov Decision Processes $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ the dynamics transition function and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ the reward function. In the following, γ denotes a discount factor. We assume that both $\mathcal{S}$ and $\mathcal{A}$ are continuous and consider a controller, or policy, $\pi_\theta : \mathcal{S} \to \mathcal{A}$, a neural network parameterized by $\theta \in \Theta$, which is called a *solution* to the problem. The *fitness* $F : \Theta \to \mathbb{R}$ of a solution measures its performance, defined as the expectation over the sum of rewards obtained by policy π_θ. A solution with high fitness is said to be *high-performing*. We introduce a behavior descriptor (BD) space $\mathcal{B}$, a behavior descriptor extraction function $\xi : \Theta \to \mathcal{B}$, and define a distance metric $||.||_\mathcal{B}$ over $\mathcal{B}$. The *diversity* of a set of K solutions $\{\theta_k\}_{k=1,\dots,K}$ is defined as $d : \Theta^K \to \mathbb{R}^+$:

$$d\left(\{\theta_k\}_{k=1,\dots,K}\right) = \sum_{i=1}^{K} \min_{k \neq i} ||\xi(\theta_i), \xi(\theta_k)||_\mathcal{B}, \tag{3.1}$$

meaning that a set of solutions is diverse if the solutions are distant with respect to each other in the sense of $||.||_{\mathcal{B}}$. Following the objective of the family of quality-diversity algorithms, we evolve a population of *diverse* and *high-performing* solutions.

3.2.2 The MAP-Elites Algorithm

MAP-Elites (see Section 2.2.2 and Algorithm 1 for a detailed introduction) is a simple yet state-of-the-art QD algorithm that has been successfully applied to a wide range of challenging problems such as robot damage recovery (Cully, Clune, Tarapore, et al., 2015), molecular robotic control (Cazenille, Bredeche, and Aubert-Kato, 2019) and game design (Alvarez, Dahlskog, Font, et al., 2019). In MAP-Elites, the behavior descriptor space $\mathcal{B}$ is discretized into a grid of cells, also called niches, with the aim of filling each cell with a high-performing solution. A variant, called CVT MAP-Elites uses Centroidal Voronoi Tesselation (Vassiliades, Chatzilygeroudis, and Mouret, 2016) to divide the grid into the desired number of cells. The algorithm starts with an empty grid and an initial random set of K solutions that are evaluated and added to the grid by following simple insertion rules. If the cell corresponding to the behavior descriptors of a solution is empty, then the solution is added to this cell. If there is already a solution in the cell, the new solution replaces it only if it has greater fitness. At each iteration, P existing solutions are sampled uniformly from the grid and randomly mutated to create P new solutions. These new solutions are then evaluated and added to the grid following the same insertion rules. This cycle is repeated until convergence or for a given budget of iterations. ME is a compelling and efficient method, but it suffers from low sample efficiency as it relies on random mutations.

3.2.3 TD3

In opposition to standard evolutionary methods that rely on random updates, RL policy gradient methods exploit the structure of the MDP to compute efficient performance-driven updates to improve the policy. Among many policy gradient algorithms, TD3 (Fujimoto, Van Hoof, and Meger, 2018) shows state-of-the-art performance in environments with continuous action space and large state space, and relies on the deterministic policy gradient update (Silver, Lever, Heess, et al., 2014) to train deterministic policies. In most policy gradients methods, a critic $Q^{\pi} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ implemented by a neural network is introduced. The critic evaluates the expected fitness of policy π when performing an episode starting from a state-action pair (s, a). The policy is updated to take the actions that will maximise the critic value estimation in each state. Both the actor and critic have an associated target network and use delayed policy updates to stabilize the training.

The TD3 algorithm adds mechanisms on top of the deterministic policy gradient update that were demonstrated to significantly improve performance and stability in practice (Fujimoto,

Van Hoof, and Meger, 2018). Namely, two critics are used together and the minimum of their values is used to mitigate overestimation issues. A Gaussian noise is added to the actions taken by the actor when computing the deterministic policy gradient to enforce smoothness between similar action values. More details on the TD3 and actor-critic formalism can be found in Section 2.3.5.

3.2.4 Policy Gradient Assisted MAP-Elites

Policy Gradient Assisted MAP-Elites (PGA-ME) (Nilsson and Cully, 2021) is a recent QD algorithm designed to evolve populations of neural networks. PGA-ME builds on top of the ME algorithm and uses policy gradient updates to create a mutation operator that drives the population towards region of high fitness. Used along genetic mutations, it has shown the ability to evolve high-performing neural networks policies where ME with genetic mutations alone failed. For this purpose, PGA-ME introduces a replay buffer that stores all the transitions experienced by policies from the population during evaluation steps, and an actor-critic couple that uses those stored transitions for training (following the TD3 method presented in 3.2.3). The critic is used at each iteration to derive the quality policy gradient estimate for half of the offspring selected from the ME grid, and is trained with a dedicated actor, called the greedy policy. This greedy policy is also updated at each iteration by the critic and is added back to the ME grid, even if its fitness is lower than individuals of similar behavior.

3.3 Diversity Policy Gradient

In this section, we introduce the Diversity Policy Gradient (D-PG), which aims at using time-step level information in order to mutate policies towards diversity. We characterize the behavior of policies as a function of the states they visit and define a diversity reward at the time-step-level that is correlated with the diversity at the episode level. This reward is computed based on distance of a state compared to the nearest states visited by other controllers in the ME grid. Maximizing the accumulated diversity rewards induces an increase of the diversity in the population we are optimizing. The following section provides a mathematical motivation for this approach.

3.3.1 Mathematical Formulation

In this section, we motivate and formally introduce the D-PG computations. Let us assume that we have a ME grid containing K solutions $(\theta_1, \dots, \theta_K)$ and that we sample θ_1 from the grid to evolve it. We aim to update θ_1 in a manner that increases the population diversity (defined in

Equation (3.1)). For this purpose, we compute the gradient of the diversity with respect to θ_1 and update θ_1 in its direction using standard gradient ascent techniques.

Proposition 3.1. *The gradient of diversity with respect to θ_1 can be written as:*

$$\begin{cases} \nabla_{\theta_1} d(\{\theta_k\}_{k=1,\dots,K}) = \nabla_{\theta_1} n(\theta_1, (\theta_j)_{2\leq j\leq J}) \\ \textit{where } \; n(\theta_1, (\theta_j)_{2\leq j\leq J}) = \sum_{j=2}^{J} ||\xi(\theta_1), \xi(\theta_j)||_{\mathcal{B}}, \end{cases} \tag{3.2}$$

where θ_2 is θ_1 closest neighbour and (θ_j) with $j = 3, \dots, J$ are the elements in the population for which θ_1 is the nearest neighbour.

We call n the novelty of θ_1 with respect to its nearest neighbor. This proposition means that we can increase the diversity of the population by increasing the novelty of θ_1 with respect to the solutions for which it is the nearest neighbor. The proof of Proposition 3.1 is as follows:

Proof. (of proposition 3.1) The diversity in Equation 3.1 can be split into three terms: the distance of θ_1 to its nearest neighbor (defined as θ_2), the distance of θ_1 to the θ_j for which θ_1 is the nearest neighbor (defined $\{\theta_j\}_{j=3,\dots,K}$ [1]) and a third term that does not depend on θ_1:

$$\begin{aligned} d(\{\theta_k\}_{k=1,\dots,K}) &= ||\xi(\theta_1), \xi(\theta_2)||_{\mathcal{B}} + \sum_{j=3}^{J} ||\xi(\theta_1), \xi(\theta_j)||_{\mathcal{B}} + M \\ &= \sum_{j=2}^{J} ||\xi(\theta_1), \xi(\theta_j)||_{\mathcal{B}} + M, \end{aligned}$$

where $M = \sum_{i\notin\{1,\dots,J\}} \min_{k\neq i} ||\xi(\theta_i), \xi(\theta_k)||_{\mathcal{B}}$ does not depend on θ_1. Hence, the diversity gradient with respect to θ_1 is:

$$\nabla_{\theta_1} d(\{\theta_k\}_{k=1,\dots,K}) = \nabla_{\theta_1} \sum_{j=2}^{J} ||\xi(\theta_1), \xi(\theta_j)||_{\mathcal{B}},$$

As the remaining term is precisely defined as the novelty n:

$$n(\theta_1, (\theta_j)_{2\leq j\leq J}) = \sum_{j=2}^{J} ||\xi(\theta_1), \xi(\theta_j)||_{\mathcal{B}},$$

We get the final relation:

$$\nabla_{\theta_1} d(\{\theta_k\}_{k=1,\dots,K}) = \nabla_{\theta_1} n(\theta_1, (\theta_j)_{2\leq j\leq J}).$$

□

[1] Remark: θ_2 can appear twice in the list $(\theta_j)_{2\leq j\leq J}$

Under this form, the diversity gradient cannot benefit from the variance reduction methods in the RL literature to efficiently compute policy gradients (Sutton, McAllester, Singh, et al., 1999). To this end, we express it as a gradient over the expectation of a sum of scalar quantities obtained by policy π_{θ_1} at each step when interacting with the environment. We introduce a novel space $\mathcal{D}$, dubbed *state descriptor space* and a *state descriptor extraction function* $\psi : \mathcal{S} \to \mathcal{D}$. We assume $\mathcal{D}$ and $\mathcal{B}$ have the same dimension. The notion of state descriptor (which is analogous to the behavior descriptor but at the timestep level rather than the trajectory level) will be used in the following to link diversity at the time step level to the diversity at the trajectory level. In this context, if the following compatibility equation is satisfied:

$$\begin{cases} n(\theta_1, (\theta_j)_{2\leq j\leq J}) = \mathbb{E}_{\pi_{\theta_1}} \sum_t n(s_t, (\theta_j)_{2\leq j\leq J}) \\ \text{where } n(s, (\theta_j)_{j=1,\dots,J}) = \sum_{j=1}^{J} \mathbb{E}_{\pi_{\theta_j}} \sum_t ||\psi(s), \psi(s_t)||_{\mathcal{D}}, \end{cases} \tag{3.3}$$

then the diversity policy gradient can be computed as:

$$\nabla_{\theta_1} d(\{\theta_k\}_{k=1,\dots,K}) = \nabla_{\theta_1} \mathbb{E}_{\pi_{\theta_1}} \sum_t n(s_t, (\theta_j)_{2\leq j\leq J}). \tag{3.4}$$

The idea behind Equation 3.3 is to link novelty defined at the solution level to a notion of novelty defined at the time step level. The information that we use at the time step level is the current state of the environment, so we describe a solution novelty as the novelty of its visited state relative to other solutions:

$$n(\theta_1, (\theta_j)_{2\leq j\leq J}) = \mathbb{E}_{\pi_{\theta_1}} \sum_t n(s_t, (\theta_j)_{2\leq j\leq J}).$$

Furthermore, we assume that a state is novel w.r.t. some solutions if it is novel w.r.t. to the states visited by these solutions:

$$n(s, (\theta_j)_{j=1,\dots,J}) = \sum_{j=1}^{J} \mathbb{E}_{\pi_{\theta_j}} \sum_t ||\psi(s), \psi(s_t)||_{\mathcal{D}}.$$

Finally, by replacing novelty at the solution level in proposition 3.1 by novelty at the state level from Equation 3.3, we get the formulation of the diversity policy gradient given in Equation 3.4. The obtained expression corresponds to the classical policy gradient setting where $\gamma = 1$ and where the corresponding reward signal, here dubbed diversity reward, is computed as $r_t^D = n(s_t, (\theta_j)_{2\leq j\leq J})$. Therefore, this gradient can be computed using any PG estimation technique replacing the environment reward by the diversity reward r_t^D. Equation (3.3) enforces a relation between $\mathcal{B}$ and $\mathcal{D}$ and between extraction functions ψ and ξ. In practice, it may be hard to define the behavior descriptor and state descriptor of a solution that satisfy this relation while being meaningful to the problem at hand and tractable. Nevertheless, a strict

equality is not necessary: a positive correlation between the two hand-sides of the equation is sufficient for diversity seeking.

3.4 Related Work

Simultaneously maximizing diversity and performance is the central goal of QD methods (Pugh, Soros, and Stanley, 2016; Cully and Demiris, 2017). Among the various possible combinations offered by the QD framework (Cully and Demiris, 2017), Novelty Search with Local Competition (NSLC) (Lehman and Stanley, 2011b) and MAP-Elites (Mouret and Clune, 2015) are the two most popular algorithms. NSLC builds on the Novelty Search (NS) algorithm (Lehman and Stanley, 2011a) and maintains an unstructured archive of solutions selected for their local performance while ME uniformly samples individuals from a structured grid that discretizes the BD space. Although very efficient in small parameter spaces, those methods struggle to scale to bigger spaces which limits their application to neuroevolution.

3.4.1 Gradients in QD

Algorithms mixing quality-diversity and evolution strategies, dubbed QD-ES, have been developed with the objective of improving the data-efficiency of the mutations used in QD thanks to evolution strategies. Algorithms such as NSR-ES and NSRA-ES have been applied to challenging continuous control environments (Conti, Madhavan, Such, et al., 2018). But, as outlined by Colas, Madhavan, Huizinga, et al. (2020), they still suffer from poor sample efficiency and the diversity and environment reward functions can be mixed in a more efficient way. ME-ES (Colas, Madhavan, Huizinga, et al., 2020) goes one step further in that direction, achieving a better mix of quality and diversity seeking through the use of a MAP-Elites grid and two specialized ES populations. Using these methods was shown to be critically more successful than population-based GA algorithms (Salimans, Ho, Chen, et al., 2017), but they rely on heavy computation and remain significantly less sample efficient than off-policy deep RL methods, as they do not leverage the analytical computation of the policy gradient at the time step level. Differentiable QD (Fontaine and Nikolaidis, 2021) improves data-efficiency in QD with an efficient search in the descriptor space but does not tackle neuroevolution and is limited to problems where the fitness and behavior descriptor functions are differentiable, which is not the case in a majority of robotic control environments.

3.4.2 Exploration and Diversity in RL

Although very efficient to tackle large state/action space problems, most reinforcement learning algorithms struggle when facing hard-exploration environments. RL methods generally seek diversity either in the state space or in the action space. This is the case of algorithms maintaining a population of RL agents for exploration without an explicit diversity criterion (Jaderberg, Dalibard, Osindero, et al., 2017) or algorithms explicitly looking for diversity in the action space rather than in the state space like ARAC (Doan, Mazoure, Durand, et al., 2019), AGAC (Flet-Berliac, Ferret, Pietquin, et al., 2021), P3S-TD3 (Jung, Park, and Sung, 2020) and DVD (Parker-Holder, Pacchiano, Choromanski, et al., 2020).

An exception is *Curiosity Search* (Stanton and Clune, 2016) which defines a notion of *intra-life novelty* that is similar to state novelty defined in Section 3.3. Our work is also related to algorithms using RL mechanisms to search for diversity only, like DIAYN (Eysenbach, Gupta, Ibarz, et al., 2018) and others (Islam, Ahmed, and Precup, 2019; Lee, Eysenbach, Parisotto, et al., 2019; Pong, Dalal, Lin, et al., 2019). These methods have proven useful in sparse reward situations, but they are inherently limited when the reward signal can orient exploration, as they ignore it. Other works sequentially combine diversity seeking and RL. The GEP-PG algorithm (Colas, Sigaud, and Oudeyer, 2018) combines a diversity seeking component, namely *Goal Exploration Processes* (Forestier, Mollard, and Oudeyer, 2017) and the DDPG RL algorithm (Lillicrap, Hunt, Pritzel, et al., 2015). This combination of exploration-then-exploitation is also present in Go-Explore (Ecoffet, Huizinga, Lehman, et al., 2019). These sequential approaches first look for diverse behaviors before optimizing performance.

3.4.3 Mixing Policy Gradient and Evolution

The fruitful synergy between evolutionary and RL methods has been explored in many recent methods, notably ERL (Khadka and Tumer, 2018), CERL (Khadka, Majumdar, Miret, et al., 2019), CEM-RL (Pourchot and Sigaud, 2018) and PGA-ME (Nilsson and Cully, 2021). ERL and CEM-RL mix evolution strategies and RL to evolve a population of agents to maximize quality but ignores the diversity of the population.

Policy Gradient Assisted MAP-Elites (PGA-ME) successfully combines QD and RL. This algorithm scales MAP-Elites to neuroevolution by evolving half of its offspring with a quality policy gradient update instead of using a genetic mutation alone. Nevertheless, the search for diversity is only explicitly done with the genetic mutation. Although based on a efficient crossover strategy (Vassiliades and Mouret, 2018), it remains a divergent search with limited data-efficiency and ignores the analytical structure of the controller. Despite having been proven effective in classic locomotion environments, our experiments show that PGA-ME

struggles on hard-exploration environments with deceptive rewards. QD-PG builds on the ideas introduced in PGA-ME but replaces the genetic mutation by the diversity policy gradient introduced in section 3.3. This data-efficient mechanism explicitly using time-step information to seek diversity helps to scale ME to neuroevolution for hard-exploration tasks. To the best of our knowledge, QD-PG is the first algorithm optimizing both diversity and performance in the solution space and in the state space, using a sample-efficient policy gradient computation method for the latter.

3.5 Methods

Our full algorithm is called QD-PG, its pseudo code is given in Algorithm 4. QD-PG is an iterative algorithm based on MAP-Elites that replaces random mutations with policy gradient updates. As we aim to improve sample efficiency by using an off-policy policy gradient method in a continuous action space, we rely on TD3 to compute policy gradients updates.QD-PG maintains three permanent structures. On the QD side, a CVT MAP-Elites grid stores the most novel and performing solutions. On the RL side, a replay buffer contains all transitions collected when evaluating solutions and an archive $\mathbb{A}$ stores all state descriptors obtained so far. QD-PG starts with an initial population of random solutions, evaluates them and inserts them into the MAP-Elites grid. At each iteration, solutions are sampled from the grid, copied, and updated. The updated solutions are then evaluated through one episode in the environment and inserted into the grid according to the usual insertion rules. Transitions collected during evaluation are stored in the replay buffer, and state descriptors are stored in the archive $\mathbb{A}$. Note that these state descriptors are first filtered to avoid insertion in the archive of multiple state descriptors that are too close to each other.

During the update step, half the population is updated with Q-PG ascent and the other half with D-PG ascent. The choice of whether an policy is updated for quality or diversity is random, meaning that it can be updated for quality and later for diversity if selected again. To justify this design, we show in Section 3.7 that updating consecutively for quality and diversity outperforms updating based on joint criteria. Both gradients are computed from batches of transitions sampled from the replay buffer. The Q-PG is computed from the usual environment rewards (similar to TD3 or PGA-ME) whereas for D-PG, we get "fresh" novelty rewards as:

$$r_t^D = \sum_{j=1}^{J} ||\psi(s_t), \psi(s_j)||_{\mathcal{D}}, \tag{3.5}$$

where $(s_j)_{j=1,\dots,J}$ are the J nearest neighbors of state s_t in the archive $\mathbb{A}$. Diversity rewards must be recomputed at each update because $\mathbb{A}$ changes during training. Following Equation (3.3), diversity rewards should be computed as the sum of the distances between the descriptor of s_t

and the descriptors of all the states visited by a list of J solutions. In practice, we consider the J nearest neighbors of s_t. This choice simplifies the algorithm, is faster to compute and works well in practice.

Algorithm 4 QD-PG

Given: Sample size N, Total number of steps max_steps, Behavior and state descriptor extraction functions ξ, ψ
Initialize: MAP-Elites grid $\mathbb{M}$, state descriptors archive $\mathbb{A}$, Replay Buffer $\mathbb{B}$, N initial actors $\{\pi_{\theta_i}\}_{i=1}^{N}$, 2 critics Q_w^D, Q_v^Q
$total\_steps \leftarrow 0$

// Main loop
while $total\_steps < max\_steps$ **do**
    if $total\_steps > 0$ **then**
        // Sampling and mutation
        Sample generation $\{\pi_{\theta_i}\}_{i=1}^{N}$ from grid $\mathbb{M}$
        Sample batches of transitions in replay buffer $\mathbb{B}$
        Compute fresh novelty rewards for half the batches
        Update half the generation for diversity
        Update the other half for quality
        Update diversity and quality critics Q_w^D and Q_v^Q

    // Evaluation and insertion in grid
    Evaluate the generation and store collected transitions in $\mathbb{B}$
    Store state descriptors in archive $\mathbb{A}$
    Update $total\_steps$ with the total number of collected steps
    Add the updated generation to the grid $\mathbb{M}$

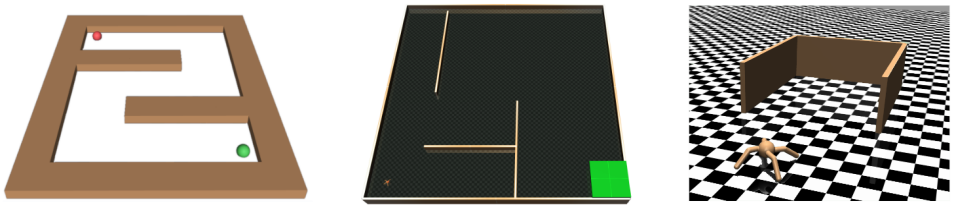
TD3 relies on a parameterized critic to reduce the variance of its policy gradient estimate. In QD-PG, we maintain two parameterized critics Q_w^D and Q_v^Q, respectively dubbed diversity and quality critics. Every time a policy gradient is computed, QD-PG also updates the corresponding critic. Both critics are trained using transitions from all policies in the population instead of just one. This particularity empirically helps to avoid local minima in exploration environments, where a unique policy could get stuck and mislead the values learned by the critic. Critic parameters are shared among the population (as in CEM Pourchot and Sigaud, 2018), reasons for doing so come from the fact that diversity is not stationary, as it depends on the current population. If each policy had its own diversity critic, since an policy may not be selected for a large number of generations before being selected again, its critic would convey an outdated picture of the evolving diversity. A side benefit of critic sharing is that both critics become accurate faster as they combine experience from all policies. Finally, as in TD3, we use pairs of critics and target critics to fight the overestimation bias.

3.6 Experiments

In this section, we first present the simulated environments and experiments that we leverage to answer the following questions: 1. Can QD-PG produce collections of diverse and high-performing neural policies and what are the advantages to do so? 2. Is QD-PG more sample efficient than its evolutionary competitors? 3. How difficult are the considered benchmarks for classical policy gradients methods? 4. What is the usefulness of the different components of QD-PG? Finally, we provide further details on the actual implementation of QD-PG and computational details of our experimental setup.

3.6.1 Environments

We assess QD-PG capabilities in continuous control environments that exhibit high dimensional observation and action spaces as well as deceptive rewards. The size of the state/action space makes exploration difficult for genetic algorithms and evolution strategies. Deceptive rewards creates exploration difficulties particularly challenging for classical RL methods. We consider three OpenAI Gym environments based on the MuJoCo physics engine that all exhibit strong deceptive rewards: Point-Maze, Ant-Maze and Ant-Trap. Such environments have also been widely used in previous works (Frans, Ho, Chen, et al., 2018; Colas, Madhavan, Huizinga, et al., 2020; Parker-Holder, Pacchiano, Choromanski, et al., 2020; Shi, Li, Zheng, et al., 2020).



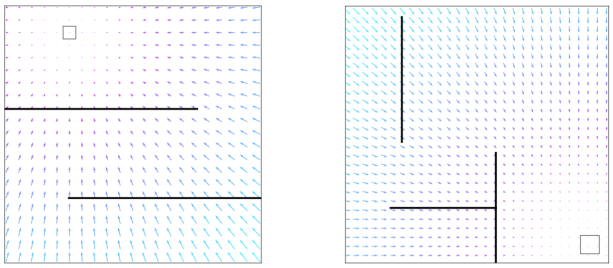
(a) Point-Maze (b) Ant-Maze (b) Ant-Trap

Figure 3.3 – Evaluation environments. The state and action spaces in Point-Maze are 2×2, whereas they are 29×8 in Ant-Maze and 113×8 in Ant-Trap.

In the Point-Maze environment, an agent represented as a green sphere must find the exit of the maze depicted in Figure 3.3a, represented as a red sphere. An observation contains the agent position at time t, and an action corresponds to position increments along the x and y axes. The reward is expressed as the negative Euclidean distance between the center of gravity of the agent and the exit center. The trajectory length cannot exceed 200 steps. The Ant-Maze environment is modified from OpenAI Gym Ant-v2 (Brockman, Cheung, Pettersson, et al., 2016) and also used in (Frans, Ho, Chen, et al., 2018; Colas, Madhavan, Huizinga, et al., 2020). In Ant-Maze, a four-legged ant starts in the bottom left of a maze and has to reach a goal zone located in the lower right part (green area in Figure 3.3b). As in Point-Maze, the

reward is expressed as the negative Euclidean distance to the center of the goal zone. The final performance is the maximum reward received during an episode. The environment is considered solved when an agent obtains a score greater than -10. An episode consists of 3000 time steps, this horizon is three times larger than in usual MuJoCo environments, making this environment particularly challenging for RL based methods (Vemula, Sun, and Bagnell, 2019). Finally, the Ant-Trap environment also derives from Ant-v2 and is inspired from Colas, Madhavan, Huizinga, et al. (2020) and Parker-Holder, Pacchiano, Choromanski, et al. (2020). In Ant-Trap, the four-legged ant initially appears in front of a trap and must bypass it to run as fast as possible in the forward direction (see Figure 3.3c). As in Ant-v2, the reward is computed as the ant velocity on the x-axis. The trap consists of three walls forming a dead-end directly in front of the ant. In this environment, the trajectory length cannot exceed 1000 steps.

In Point-Maze, the state and action spaces are two-dimensional. By contrast, in Ant-Maze and Ant-Trap, the number of dimensions of their observation spaces are respectively equal to 29 and 113 while the number of dimensions of their action spaces are both equal to 8, making these two environments much more challenging as they require larger controllers. The Ant-Trap environment also differs from mazes as it is open-ended, i.e., the space to be explored by the agent is unlimited, unlike mazes where this space is restricted by the walls. In this case, a state descriptor corresponds to the ant position that is clipped to remain in a given range. On the y-axis, this range is defined as three times the width of the trap. On the x-axis, this range begins slightly behind the starting position of the ant and is large enough to let it accelerate along this axis.



(a) Point-Maze (b) Ant-Maze

Figure 3.4 – Gradient maps on Point-Maze and Ant-Maze. Black lines are maze walls, arrows depict gradient fields and the square indicates the maze exit. Both settings present deceptive gradients as naively following them leads into a wall.

In all environments, state descriptors $\psi(s_t)$ are defined as the agent position at time step t and behavior descriptors $\xi(\theta)$ are defined as the agent position at the end of a trajectory. Therefore, we have $\mathcal{B} = \mathcal{D} = \mathbb{R}^2$, $\psi(s_t) = (x_t, y_t)$ and $\xi(\theta) = (x_T, y_T)$ where T is the trajectory length. We also take $||.||_{\mathcal{B}}$ and $||.||_{\mathcal{D}}$ as Euclidean distances. This choice does not always satisfy Equation (3.3) but is convenient in practice and led to satisfactory results. The peculiarity of Ant-Trap lies in the fact that the reward is expressed as the forward velocity of the ant, thus

making the descriptors not totally aligned with the task. Figure 3.4 highlights the deceptive nature of the Point-Maze and the Ant-Maze objective functions by depicting gradient fields in both environments.

3.6.2 Baselines and Ablations

To answer question 1, we inspect the collection of solutions obtained with QD-PG and try to adapt to a new objective with a limited budget of 20 evaluations. QD-PG is then compared to three types of methods. To answer question 2, we compare QD-PG to a family of QD baselines, namely ME-ES, NSR-ES, NSRA-ES (Colas, Madhavan, Huizinga, et al., 2020). Talbe 3.1 recaps the properties of all these methods. Then, to answer question 3, we compare QD-PG to a family of policy gradient baselines. Random Network Distillation (RND) (Burda, Edwards, Storkey, et al., 2018) and Adversarially Guided Actor Critic (AGAC) (**flet-berliac2021adversarially**) add curiosity-driven exploration process to a RL agent. We also compare to the population-based RL methods CEM-RL, P3S-TD3 and to the closest method to ours, PGA-ME. Finally, to answer question 4, we propose to investigate the following matters: Can we replace alternating quality and diversity updates by a single update that optimizes for the sum of both criteria? Are quality (resp. diversity) gradients updates alone enough to fill the MAP-Elites grid? Consequently, we consider the following ablations of QD-PG: QD-PG Sum, which computes a gradient to optimize the sum of the quality and diversity rewards; D-PG, which applies diversity gradients only; and Q-PG which applies quality gradients only. Note that both D-PG and Q-PG ablations still use QD selection. For all experiments, we limited the number of seeds to 5 due to limited compute capabilities. Table 3.1 summarizes the different components present in QD-PG, its ablations and all baselines.

Table 3.1 – Ablations and baselines summary. Selec. stands for selection. The last column assesses whether the method optimizes for a collection instead of a single solution.

	Algorithm	QPG	DPG	Q Selec.	D Selec.	Collection
Ablations	QD-PG	✓	✓	✓	✓	✓
	QD-PG Sum	✓	✓	✓	✓	✓
	D-PG	X	✓	✓	✓	✓
	Q-PG	✓	X	✓	✓	✓
PG	SAC	✓	X	X	X	X
	TD3	✓	X	X	X	X
	RND	✓	✓	X	X	X
	CEM-RL	✓	X	✓	X	✓
	P3S-TD3	✓	✓	X	X	✓
	AGAC	✓	✓	X	X	X
	DIAYN	X	✓	X	X	X
	PGA-ME	✓	X	✓	✓	✓
QD	ME-ES	X	X	✓	✓	✓
	NSR-ES	X	X	✓	✓	✓
	NSRA-ES	X	X	✓	✓	✓

3.6.3 Implementation Details

We consider populations of $N = 4$ actors for the Point-Maze environment and $N = 10$ actors for Ant-Maze and Ant-Trap. We use 1 CPU thread per actor and parallelization is implemented with the Message Passing Interface (MPI) library. Our experiments are run on a standard computer with 10 CPU cores and 100 GB of RAM, although the maximum RAM consumption per experiment at any time never exceeds 10GB due to an efficient and centralized management of the MAP-Elites grid which stores all solutions. An experiment on Point-Maze typically takes between 2 and 3 hours while an experiment on Ant-Maze or Ant-Trap takes about 2 days. Note that these durations can vary significantly depending on the type of CPU used. We did not use any GPU. Computational costs of QD-PG mainly come from backpropagation during the update of each agent, and to the interaction between agents and the environment. These costs scale linearly with the population size but, as many other population-based methods, the structure of QD-PG lends itself very well to parallelization. We leverage this property and parallelize our implementation to assign one agent per CPU thread. Memory consumption also scales linearly with the number of agents. To reduce this consumption, we centralize the MAP-Elites grid on a master worker and distribute data among workers when needed. With these implementation choices, QD-PG only needs a very accessible computational budget for all experiments.

Table 3.2 – QD-PG Hyper-parameters: Ant-Maze and Ant-Trap hyper-parameters are identical and grouped under the Ant column

Parameter	**PointMaze**	**Ant**
TD3		
Optimizer	SGD	SGD
Learning rate	6×10^{-3}	3×10^{-4}
Discount factor γ	0.99	0.99
Replay buffer size	10^6	5×10^5
Hidden layers size	64/32	256/256
Activations	ReLU	ReLU
Minibatch size	256	256
Target smoothing coeff.	0.005	0.005
Delay policy update	2	2
Target update interval	1	1
Gradient steps ratio	4	0.1
State Descriptors Archive		
Archive size	10000	10000
Threshold of acceptance	0.0001	0.1
K-nearest neighbors	10	10
MAP-Elites		
Nb. of bins per dimension	5	7

In practice, to compute diversity rewards presented from Equation 3.5, we maintain a FIFO archive $\mathbb{A}$ of the state descriptors encountered so far. When a transition $(s_t, a_t, r_t, s_{t+1}, \psi(s_t))$ is stored in the replay buffer, we also add $\psi(s_t)$ to $\mathbb{A}$. We only add a state descriptor in $\mathbb{A}$ if its mean Euclidean distance to its K nearest neighbors is greater than an acceptance threshold. This filtering step enables to keep the archive size reasonable and to facilitate the computation of the K nearest neighbors. The values of K and of the threshold are given in Table 3.2. When a batch of transitions is collected during the update phase, we recompute fresh diversity rewards r_t^D as the mean Euclidean distance between the sampled state descriptors $\psi(s_t)$ and their K nearest neighbors in $\mathbb{A}$. These diversity rewards are used in place of standard rewards in sampled transitions $(s_t, a_t, r_t^D, s_{t+1}, \psi(s_t))$ to compute the diversity policy gradient.

3.7 Results

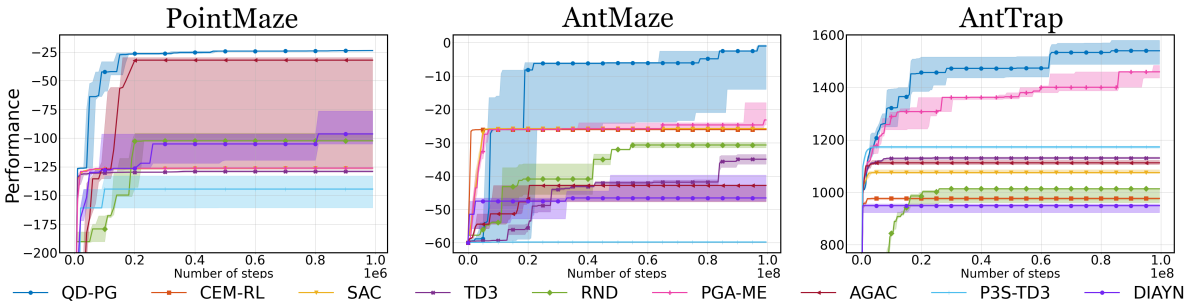


Figure 3.5 – Performance of QD-PG and baseline algorithms on all environments. Plots present median bounded by first and third quartiles.

Can QD-PG produce collections of neural policies and what are the advantages to do so? Table 3.4 and figure 3.5 present QD-PG performances. In all environments, QD-PG manages to find working solutions that avoid local minima and reach the overall objective. In addition to its exploration capabilities, QD-PG generates collections of high performing solutions in a single run. During the Ant-Trap experiment, the final collection of solutions returned by QD-PG contained, among others, 5 solutions that were within a 10% performance margin from the best one. As illustrated in Figure 3.1, these policies typically differ in their gaits and preferred trajectories to circumvent the trap.

Generating a collection of diverse solutions comes with the benefit of having a repertoire of diverse solutions that can be used as alternatives when the MDP changes (Cully, Clune, Tarapore, et al., 2015). To show that QD-PG is more robust than conventional policy gradient methods and demonstrate the benefit of having a collection of solutions, we propose a fast-adaptation experiment where the reward signal of the Ant-Maze environment is randomly changed.

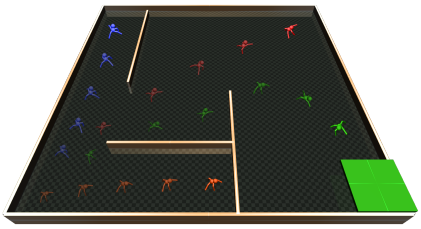


Figure 3.6 – QD-PG produces a collection of diverse solutions. In Ant-Maze, even after setting new randomly located goals, the ME grid still contains solutions that are suited for the new objectives.

We replace the original goal in the bottom right part of the maze (see Figure 3.6) with a new randomly located goal in the maze. Instead of running QD-PG to optimize for this new objective, we run a Bayesian optimization process to quickly find a good solution among the ones already stored in the grid. With a budget of only 20 solutions to be tested during the Bayesian optimization process, we are able to quickly recover a good solution for the new objective. We repeat this experiment 100 times, each time with a different random goal, and obtain an average performance of -10 with a standard deviation of 9. In other words, 20 interaction episodes (corresponding to 60.000 time steps) suffice for the adaptation process to find a solution that performs well for the new objective without the need to re-train agents.

Browsing the MAP-Elites grid in an exhaustive way —instead of relying on Bayesian search— is another option to find a good solution for a new objective. However, the number of solutions to be tested with this option increases quadratically w.r.t. the number of meshes used to discretize the dimensions of the BD space. As shown in Table 3.2, we used a 7×7 grid to train QD-PG in the Ant-Maze environment, containing a maximum of 49 solutions. In this setting, the difference in computation cost between exhaustive search and Bayesian optimization is negligible. To ensure that fast adaptation scales to finely discretized MAP-Elites grids, we retrained QD-PG and reproduced this experiment with a 100×100 grid, containing thousands of solutions, and obtained an average performance of -9 with a standard deviation of 7 (this time with a budget of 50 solutions to be tested during the Bayesian search process).

Figure 3.7 maps these 100 fast adaptation experiments to their respective goal location and performance. In each square, we display the score of the best experiment whose goal was sampled in this region of the maze. For instance, the square in the top left corner of the map corresponds to one of the 100 fast adaptation experiments that sampled its goal in this part of the maze, and obtained a performance of -12 after testing 50 solutions from the MAP-Elites grid during the Bayesian optimization process. Empty squares correspond to regions of the maze where no goals were sampled during the 100 experiments.

Is QD-PG more sample efficient than its evolutionary competitors? Table 3.3 compares QD-PG to deep neuroevolution algorithms with a diversity seeking component (ME-ES, NSR-ES, NSRA-ES, CEM-RL) in terms of sample efficiency. QD-PG runs on 10 CPU cores for 2 days while its competitors used 1000 CPU cores for the same duration. Nonetheless, QD-PG matches the asymptotic performance of ME-ES using two orders of magnitude fewer samples, explaining the lower resource requirements.

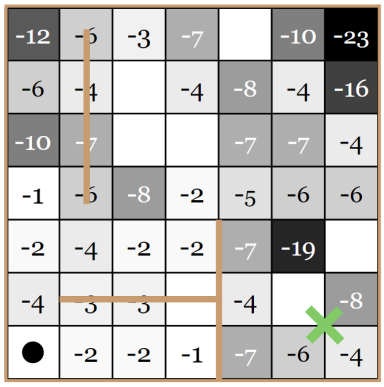


Figure 3.7 – Results of the fast-adaptation experiment repeated 100 times in Ant-Maze. In each square, we display the score of the best experiment whose goal was sampled in this region of the maze, as several experiments may have goals in the same square. Empty squares correspond to regions where no goal was sampled during the experiments. The black circle shows the agent starting position, the black lines represent the walls in Ant-Maze and the green cross marks the location of the original goal in Ant-Maze.

We see three reasons for the improved sample efficiency of QD-PG: 1) QD-PG leverages a replay buffer and can re-use each sample several times. 2) QD-PG leverages novelty at the state level and can exploit all collected transitions to maximize quality and diversity. For instance, in Ant-Maze, a trajectory brings 3000 samples to QD-PG while standard QD methods would consider it a unique sample. 3) PG exploits the analytical gradient between the neural network weights and the resulting policy action distribution and estimates only the impact of the distribution on the return. By contrast, standard QD methods directly estimate the impact on the return of randomly modifying the weights.

Table 3.3 – Comparison to evolutionary competitors on Ant-Maze. The **Ratio** column compares the sample efficiency of a method to QD-PG.

Algorithm	Final Perf.	Steps to goal	Ratio
QD-PG	$-1(\pm 7)$	**8.4e7**	**1**
CEM-RL	$-26(\pm 0)$	∞	∞
ME-ES	$-5(\pm 0)$	$2.4e10$	286
NSR-ES	$-26(\pm 0)$	∞	∞
NSRA-ES	$\mathbf{-2}(\pm \mathbf{1})$	$2.1e10$	249

How challenging are the considered benchmarks for policy gradients methods? Table 3.4 compares QD-PG to state-of-the-art policy gradient algorithms and validates that classical policy gradient methods fail to find optimal solutions in deceptive environments. TD3 quickly converges to local minima of performance resulting from being attracted in dead-ends by the deceptive gradients. While we may expect SAC to better explore due to entropy regularization, it also converges to that same local minima. Besides, despite its exploration mechanism based

on CEM, CEM-RL also quickly converges to local optima in all benchmarks, confirming the need for a dedicated diversity seeking component. RND, which adds an exploration bonus used as an intrinsic reward, also demonstrates performances inferior to QD-PG in all environments but manages to solve Point-Maze. In Ant-Trap, RND extensively explores the BD space but fails to obtain high returns. Although maintaining diversity into a population of reinforcement learning agents, AGAC and P3S-TD3 are not able to explore enough of the environments to solve them, showing some limits of exploration through the action space.

Table 3.4 – Final performance compared to ablations and PG baselines. Performance is measured as the minimum distance to the goal in Ant-Maze (10^8 steps) and as the episode return in Point-Maze (10^6 steps) and Ant-Trap (10^8 steps).

Algorithm	Point-Maze	Ant-Maze	Ant-Trap
QD-PG	$\mathbf{-24}(\pm\mathbf{0})$	$\mathbf{-1}(\pm\mathbf{7})$	$\mathbf{1540}(\pm\mathbf{46})$
QD-PG Sum	$-24(\pm0)$	$-33(\pm4)$	$1013(\pm0)$
D-PG	$-36(\pm2)$	$-59(\pm0)$	$1014(\pm0)$
Q-PG	$-128(\pm0)$	$-59(\pm0)$	$1139(\pm38)$
SAC	$-126(\pm0)$	$-26(\pm0)$	$1075(\pm7)$
TD3	$-129(\pm1)$	$-35(\pm1)$	$1131(\pm4)$
RND	$-102(\pm4)$	$-31(\pm1)$	$1014(\pm27)$
CEM-RL	$-312(\pm1)$	$-26(\pm0)$	$977(\pm3)$
P3S-TD3	$-144(\pm14)$	$-60(\pm0)$	$1173(\pm4)$
AGAC	$-32(\pm49)$	$-43(\pm3)$	$1113(\pm8)$
DIAYN	$-96(\pm14)$	$-47(\pm4)$	$949(\pm34)$
PGA-ME	$-126(\pm0)$	$-18(\pm6)$	$1455(\pm17)$

DIAYN is able to explore in Point-Maze but does not fully explore the environment and hence does not reach the goal before the end of the 10^6 steps. DIAYN ensures that its skills explore states different enough to be discriminated, but once they are, there is no incentive to further explore. Moreover, DIAYN in Ant-Trap shows a limit of unsupervised learning methods: Ant-Trap is an example of environment where the behavior descriptor chosen is not perfectly aligned with the performance measured. As rewards are completely ignored by DIAYN, the produced controllers have very low fitness.

Figure 3.5 relates performance of QD-PG and PGA-ME along time on Point-Maze, Ant-Maze, Ant-Trap and shows that QD-PG outperforms the latter both in final performance and data-efficiency by a significant margin. It was the first time that PGA-ME was assessed in exploration environments. This definitely confirms that the Genetic mutation used by PGA-ME struggles in big parameter space while the Diversity Policy Gradient offers a data-efficient diversity-seeking mechanism to improve the exploration of MAP-Elites in high-dimensional deceptive environments. We tried QD-PG on the locomotion tasks from the original paper (Nilsson and Cully, 2021) and found it hard to design state descriptor extractors Ψ such that our diversity rewards would be strongly correlated with the behavior descriptor ξ used in PGA-ME

(Nilsson and Cully, 2021) in the sense of equation 3.3, showing a limitation of our work. This leads us to the conclusion that keeping genetic mutations in addition to our diversity policy gradient could end up in a fruitful and robust synergy. Finally, we observed that the learning process of QD-PG on Ant-Maze sometimes struggled to start, explaining the high interquartile distance of performances on Ant-Maze in Figure 3.5).

What is the usefulness of the different components of QD-PG ? The ablation study in Table 3.4 shows that when maximizing quality only, Q-PG fails due to the deceptive nature of the reward and when maximizing diversity only, D-PG sufficiently explores to solve Point-Maze but requires more steps and finds lower-performing solutions. When optimizing simultaneously for quality and diversity, QD-PG Sum fails to learn in Ant-Trap and manages to solve the task in Ant-Maze but requires more samples than QD-PG. We hypothesize that quality and diversity rewards may give rise to conflicting gradients. Therefore, both rewards cancel each other, preventing any learning. This study validates the usefulness of QD-PG components: 1. optimizing for diversity is required to overcome the deceptive nature of the reward, 2. adding quality optimization provides better asymptotic performance, 3. it is better to disentangle quality and diversity updates.

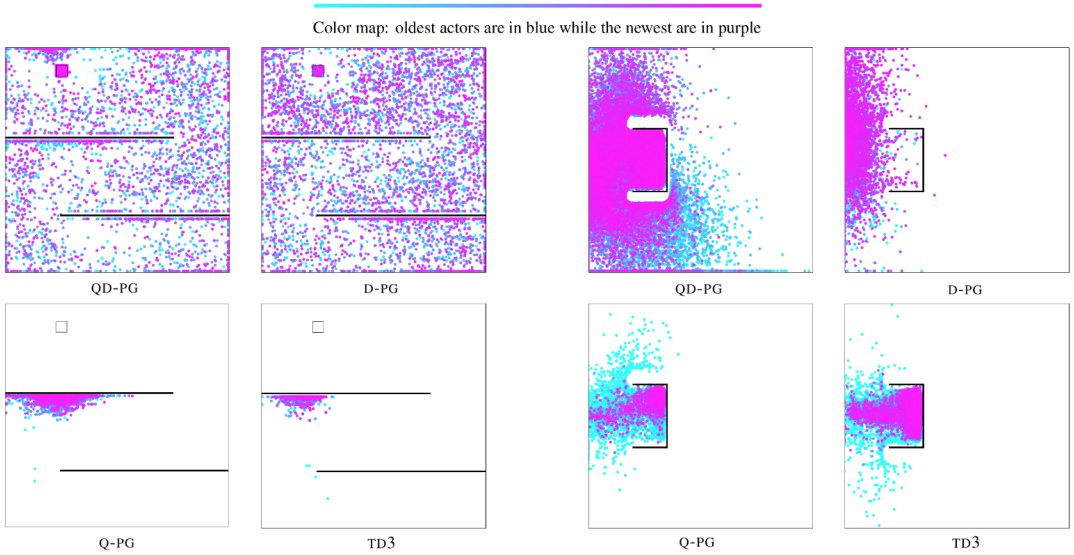


Figure 3.8 – Coverage map of the Point-Maze (left) and Ant-Trap (right) environments for all ablations. Each dot corresponds to the final position of a policy through the training process.

Figure 3.8 (left) shows coverage maps of the Point-Maze environment obtained with one representative seed by the different algorithms presented in the ablation study. A dot in the figure corresponds to the final position of an agent after an episode. The color spectrum highlights the course of training: policies evaluated early in training are in blue while newer ones are represented in purple. QD-PG and D-PG almost cover the whole BD space including the objective. Unsurprisingly, Q-PG and TD3 present very poor coverage maps, both algorithms

optimize only for quality and the MAP-Elites selection mechanism in Q-PG contributes nothing in this setting. By contrast, algorithms optimizing for diversity (QD-PG and D-PG) find the maze exit. However, as shown in Table 3.4, QD-PG which also optimizes for quality, is able to refine trajectories through the maze and obtains significantly better performance. Figure 3.8 (right) depicts the coverage maps of the Ant-Trap environment by QD-PG and TD3. Only QD-PG is able to bypass the trap and to cover a large part of the BD space.

3.8 Conclusion

This chapter introduced QD-PG, a hybrid algorithm that builds upon MAP-Elites algorithm and uses methods inspired from reinforcement learning literature to produce collections of diverse and high-performing neural policies in a sample-efficient manner. We showed experimentally that QD-PG creates multiple solutions achieving high returns in continuous control hard-exploration problems. Although more efficient than genetic mutations, the diversity policy gradient requires a state descriptor ensuring a correlation between state diversity and global diversity, which can be difficult to define when the behavior space is more complex.

Chapter 4

Generating Behavior-Conditioned Trajectories with Decision Transformers

An expert is a person who has found out by his own painful experience all the mistakes that one can make in a very narrow field.

Niels Bohr.

A natural goal induced by the creation of a quality-diversity repertoire is trying to achieve behaviors on demand, which can be done by running the corresponding policy from the repertoire. However, in uncertain environments, two problems arise. First, policies can lack robustness and repeatability, meaning that multiple episodes under slightly different conditions often result in very different behaviors. Second, due to the discrete nature of the repertoire, solutions vary discontinuously. In this chapter, we present a new approach to achieve behavior-conditioned trajectory generation on demand with high accuracy.

Contents

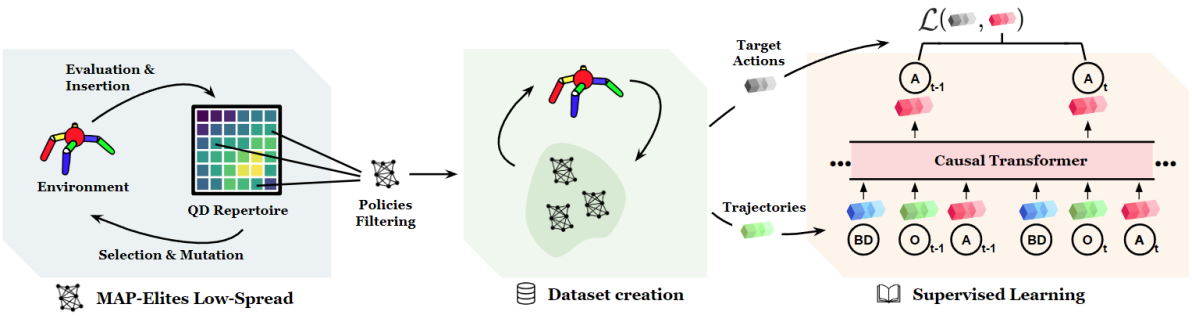


Figure 4.1 – High-level view of the Quality-Diversity Transformer pipeline. Our full algorithm is composed of three distinct and successive parts: 1. Execution of MAP-Elites Low-Spread, our variation of MAP-Elites that produces stable and consistent policies, 2. Creation of an offline dataset of trajectories by playing episodes with some of the policies produced by MAP-Elites Low-Spread, 3. Supervised training of a behavior-conditioned causal Transformer producing on-demand behaviors with high accuracy.

4.1 Introduction

In many real-world scenarios, it is important to explore the entire solution space and find diverse efficient solutions that can be used as alternatives in case a single solution fails (Cully, Clune, Tarapore, et al., 2015). More importantly, reproducing the behavior of a policy with accuracy and consistency is crucial, as it allows engineers and designers to anticipate possible failures and mitigate them with backup options. Previous work has shown that QD methods are suitable for neuroevolution (Rakicevic, Cully, and Kormushev, 2021; Flageat, Lim, Grillotti, et al., 2022) in complex, uncertain domains such as robotic manipulation and locomotion (Colas, Madhavan, Huizinga, et al., 2020; Morrison, Corke, and Leitner, 2020; Nilsson and Cully, 2021). However, they demonstrate important difficulties in producing policies that are consistent in the behavior space when the initial conditions of an episode change slightly (Flageat, Chalumeau, and Cully, 2022). For instance, solutions stored in the MAP-Elites grid of QD-PG (see Chapter 3) may have been lucky during the only evaluation episode they played. Since solutions are stored when they obtain superior fitness than the corresponding solution in the grid, this single evaluation mechanism encourages high-variance solutions that *sometimes* outperform more consistent ones. Still, *sometimes* is enough to populate a QD repertoire with high-variance solutions; during the training process, many different solutions are created and evaluated, and all it takes is one (un)lucky accident per cell of the grid to store a high-variance solution that will prevent more consistent ones to be retained.

Moving to another topic related to this chapter, recent advances in machine learning have led to the emergence of the Transformer (Vaswani, Shazeer, Parmar, et al., 2017) as a powerful and prevalent model architecture to address various problems, including text generation (Brown,

Mann, Ryder, et al., 2020), image processing (Carion, Massa, Synnaeve, et al., 2020; Dosovitskiy, Beyer, Kolesnikov, et al., 2020) and sequential decision making (Lee, Nachum, Yang, et al., 2022; Reed, Zolna, Parisotto, et al., 2022). In particular, the Decision Transformer (DT) (Chen, Lu, Rajeswaran, et al., 2021) performs conditioned sequential decision making in simulated robotics environments by leveraging supervised learning on datasets of trajectories generated by reinforcement learning (RL) policies. Unlike other approaches, its specificity is to condition its decision making process on a desired return to be obtained at the end of an episode. After an offline training phase, the DT is able to condition on a target return and to play episodes that achieve this target return with high accuracy, and even to generalize to returns that were not seen during training.

Contributions: In this work, we experimentally show that MAP-Elites-based algorithms have considerable difficulty in producing consistent policies in uncertain domains. We introduce a new metric, the policy *spread*, which measures the consistency of policies in the BD space and propose a variant of MAP-Elites, called MAP-Elites Low-Spread (ME-LS), which drives the search process towards consistent policies by selecting them for their higher fitness and lower spread. We then introduce the Quality-Diversity Transformer (QDT), a behavior-conditioned model inspired by the DT that learns to achieve behaviors on demand and leverages supervised learning on datasets of trajectories generated by repertoires of policies. We run experiments in simulated robotic environments based on the physics engine Brax (Freeman, Frey, Raichuk, et al., 2021) and show experimentally that: 1. ME-LS produces repertoires of consistent policies that replicate their behavior descriptors (BDs) over multiple episodes with varying initial conditions, 2. The QDT compresses a whole repertoire into a single policy and demonstrates even better accuracy than ME-LS policies, 3. Finally, the QDT is capable, to a certain extent, to generalize to unseen behaviors. All the code in this chapter was created using the QDax framework (Chalumeau, Lim, Boige, et al., 2024).

4.2 Related Work

Several works have investigated the behavior and fitness estimation problem in uncertain domains. Cully and Demiris (2018) aggregate multiple runs to evaluate a single solution. They perform insertion in the repertoire based on the average fitness over all episodes and based on the geometric median of all obtained BDs. Following the terminology introduced in Flageat, Chalumeau, and Cully (2022), we name this method MAP-Elites-sampling and remark that, without having been named explicitly, a very similar approach has been used in Engebraaten, Moen, Yakimenko, et al. (2020). We argue that MAP-Elites-sampling suffers from two issues: First, its focus is on finding better approximation of the true characteristics of solutions, rather than directly searching for more robust ones. Second, using the geometric mean over all BDs can result in a situation where a solution is qualified by a BD that it never

actually achieves, which could prevent another, more adequate solution to be inserted in the repertoire. To support these claims, we include ME-Sampling as a baseline in our experiments. Deep-Grid (Flageat and Cully, 2020) is a variant of MAP-Elites that employs an archive of similar previously encountered solutions to estimate the characteristics of a solution. However, Flageat, Chalumeau, and Cully (2022) show that Deep-Grid fails to find reproducible solutions as efficiently as MAP-Elites-sampling, and hypothesizes that because the method uses neighbours in the BD-space to approximate the true characteristics of solutions, it does not perform well in high dimensional search spaces –which are typical in neuroevolution tasks– where the complex relation between genotypes and BDs can be confusing for this neighbourhood-based mechanism. Adaptive-Sampling (Justesen, Risi, and Mouret, 2019) is a method that discards solutions that are evaluated too many times outside of their main cells to keep only the most reproducible solutions. Flageat and Cully (2023) propose an extensive study of the reproducibility problem, they compare existing methods (including MAP-Elites Sampling, Deep-Grid and Adaptive-Sampling) and introduce new variations. Still, as opposed to ME-LS, all these methods do not act directly on insertion but progressively measure the reproducibility of solutions and eliminate the least reproducible ones.

Algorithm 5 PGA-MAP-Elites

Given Max iteration number I, Sample size N, MAP-Elites repertoire $\mathbb{M}$, Replay Buffer $\mathbb{B}$, A critic network Q_v
Initialization: Create N random policies $\{\pi_{\theta_i}\}_{i=1}^{N}$ ▷ Initialize the system
Evaluate and insert them in $\mathbb{M}$
$iteration\_number \leftarrow 0$
while $iteration\_number < I$ **do** ▷ Main loop
    Sample N policies $\{\pi_{\theta_i}\}_{i=1,N}$ in repertoire $\mathbb{M}$ ▷ Sampling and mutation
    Mutate half the policies using the TD3 update using Q_v
    Mutate the other half with random genetic mutations
    ▷ Train the critic
    Sample batches of transitions in replay buffer $\mathbb{B}$
    Update the critic Q_v using TD3
    ▷ Evaluation
    Evaluate each new policy over 1 trajectory and store all transitions in buffer $\mathbb{B}$
    Compute their fitnesses and BDs
    ▷ Insertion in repertoire
    for each new policy **do**
        if fitness of new policy is higher than the corresponding policy in $\mathbb{M}$ **then**
            Insert new policy in $\mathbb{M}$
    $iteration\_number \leftarrow iteration\_number + 1$

Policy Gradient Assisted MAP-Elites (PGA-ME) (Nilsson and Cully, 2021) is a state-of-the-art QD algorithm bridging the gap between evolutionary and policy gradient (PG) methods. It builds upon MAP-Elites and introduces a variation operator based on policy gradient reinforcement learning to optimize for fitness. PGA-MAP-Elites uses a replay buffer to store the transitions experienced by policies from the population during evaluation steps, and also

employs an actor-critic model, using the TD3 algorithm (Fujimoto, Van Hoof, and Meger, 2018) to train on the stored transitions. The critic is utilized at each iteration to calculate the policy gradient estimate for half of the offspring selected from the MAP-Elites repertoire. A separate actor, known as the greedy controller, is also trained with the critic. This greedy controller is updated at each iteration and added to the MAP-Elites repertoire, even if its fitness is lower than other individuals with similar behavior. In this work, we replicate all ME-based experiments and adapt them to PGA-ME to show that the stated problem and our conclusions are still valid in the presence of PG variations. Algorithm 5 contains the pseudocode for PGA-ME.

The Transformer (Vaswani, Shazeer, Parmar, et al., 2017) is a popular model architecture that was specifically designed for natural language processing (NLP) tasks such as language translation (Devlin, Chang, Lee, et al., 2018), but has since been applied to a variety of other tasks, including image and speech recognition (Dong, Xu, and Xu, 2018; Dosovitskiy, Beyer, Kolesnikov, et al., 2020), text generation (Brown, Mann, Ryder, et al., 2020) and sequence modeling for RL (Chen, Lu, Rajeswaran, et al., 2021). The key innovation of the Transformer is its use of self-attention mechanism, which allows the model to efficiently weigh the importance of different parts of the input when making predictions, rather than relying solely on the order of the input as in previous architectures. Authors of the Decision Transformer (DT) (Chen, Lu, Rajeswaran, et al., 2021) propose a new approach to sequential decision making problems formalized as reinforcement learning using the Transformer architecture. They train the DT on collected experience (datasets of trajectories) using a sequence modeling objective —allowing for a more direct and effective way of performing credit assignment— and show that it matches or exceeds the performance of state-of-the-art model-free offline RL algorithms. They use a trajectory representation that allows a causal Transformer based on the GPT architecture (Radford, Narasimhan, Salimans, et al., 2018) to autoregressively model trajectories using a causal self-attention mask. Put simply, the DT predicts actions by paying attention to all prior elements in a trajectory. With this work, the authors bridge the gap between sequence modeling and reinforcement learning, proving that sequence modeling can serve as a robust algorithmic paradigm for sequential decision making.

In line with the Decision Transformer, the Trajectory Transformer (Janner, Li, and Levine, 2021) achieves goal-conditioned trajectory generation in 2D MiniGrid environments, and authors of Multi-Game Decision Transformers (Lee, Nachum, Yang, et al., 2022) train a single model that plays up to 46 Atari games with near-human performance. Even more impressively, GATO (Reed, Zolna, Parisotto, et al., 2022) is a generalist Transformer-based agent tackling a wide variety of tasks including, among others, real and simulated robotic manipulations. A point of divergence between these works and ours is the tokenization scheme used: while they tokenize each dimension of each element given as input to their models, we follow the DT scheme and directly feed the continuous observations, actions and conditioning BD to the QDT. This has the advantage of drastically reducing the computing time and resources required as

the model itself is smaller and does not grow as a function of the input dimensionality. Finally, Jegorova, Doncieux, and Hospedales (2020) introduces a behavior-conditioned generative model which generates parameters of policies that are conditioned to achieve a target behavior. However, their model is not suited for deep neuroevolution and focuses on generating simple policies consisting of dozens of parameters.

4.3 Problem Statement

Following Section 2.3.1, we consider sequential decision-making problems expressed as Markov Decision Processes (MDPs) defined by $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ the transition function and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ the reward function. We assume that both $\mathcal{S}$ and $\mathcal{A}$ are continuous and that both $\mathcal{P}$ and $\mathcal{R}$ are deterministic functions. Policies $\pi_\theta : \mathcal{S} \to \mathcal{A}$ are assumed to be implemented by neural networks parameterized by $\theta \in \Theta$, which are called *solutions* to the problem. We denote by $\tau_{[\pi_\theta, s]} \in \Omega$ a trajectory of the policy π_θ starting from the initial state s.

Contrary to previous related works (Nilsson and Cully, 2021; Chalumeau, Boige, Lim, et al., 2022; Pierrot, Macé, Chalumeau, et al., 2022; Pierrot, Richard, Beguir, et al., 2022) and due to the fact that we exclusively consider uncertain continuous control environments where the initial state is randomly sampled, we do not expect a direct mapping from a solution θ to its fitness nor to its behavior descriptor and rather consider fitnesses and behavior descriptors of *trajectories* played by π_θ. The fitness function $F : \Omega \to \mathbb{R}$ takes a trajectory $\tau_{[\pi_\theta, s]}$ as input and measures its performance, defined as the sum of rewards obtained by policy π_θ during an episode starting from initial state s. It is used to estimate the actual fitness of the solution θ, which can be theoretically defined as the expected value of the fitness for a given initial state distribution. We also introduce a behavior descriptor space $\mathcal{B}$ and a behavior descriptor extraction function $\xi : \Omega \to \mathcal{B}$ that maps a trajectory $\tau_{[\pi_\theta, s]}$ to its behavior descriptor $\xi(\tau_{[\pi_\theta, s]})$. Assuming that $\text{dist}_\mathcal{B}$ is a distance metric over $\mathcal{B}$, we define the *spread* of K trajectories as the mean distance between all pairs of behavior descriptors. We use it to estimate the spread of the solution θ (cf. Equation 4.1), which again can be theoretically defined as the expected value of the distance between the behavior descriptors of two trajectories for a given initial state distribution. In most cases, we could also use the standard distance deviation or dispersion, defined as the mean distance to the mean behavior descriptor, which has better computational complexity, but requires the definition of the mean behavior descriptor, which is trivial when they belong to a vector space, but can be difficult in the general case. Furthermore, the difference in computational complexity has a negligible impact on our method, as we always consider a limited number of evaluations for each policy.

$$\psi(\theta) \hat{=} \min_{1 \leq i < j \leq K} \text{dist}_{\mathcal{B}} \left(\xi(\tau_{[\pi_\theta, s_i]}), \xi(\tau_{[\pi_\theta, s_j]}) \right), \tag{4.1}$$

where the states $\{s_i\}_{i=1,\dots,K}$ are randomly initialized.

Put simply, the spread of a solution measures its tendency to obtain different behavior descriptors when playing multiple episodes with varying initial states. The lower the spread, the more consistent the policy is in the BD space. Note that in the experiments presented in chapter, $\mathcal{B}$ is a Euclidean space and we use the Euclidean distance over $\mathcal{B}$ as $\text{dist}_{\mathcal{B}}$.

4.3.1 Environments

The set of tasks we study is based on the Brax suite (Freeman, Frey, Raichuk, et al., 2021), a physics engine for large scale rigid body simulation written in Jax (Bradbury, Frostig, Hawkins, et al., 2018). It is worth mentioning that all policies used in this work (including the QDT and policies produced by QD algorithms) are deterministic, thus the stochasticity is only the result of the variability in the initial states, which are randomly sampled from a Gaussian distribution.

(a) Ant-Omni (b) HalfCheetah-Uni

Figure 4.2 – Illustration of the benchmark tasks used in this work, based on the Brax physics engine.

We evaluate the capacity of our method to perform in continuous control locomotion tasks that feature substantially distinct behavior spaces, as well as high dimensional observation and action spaces. In these tasks, the challenge is to move legged robots by applying torques to their joints via actuators. These types of environments are particularly challenging for evolutionary algorithms as they typically necessitate a significant number of interactions with the environment to develop high-performing policies. We follow the terminology introduced in Chalumeau, Boige, Lim, et al. (2022) to name environments and distinguish between omnidirectional and unidirectional environments by providing them with distinct names. All episodes in both environments have a maximum length of 250 timesteps.

Ant-Omni is an exploration-oriented environment in which a four-legged ant robot must move on the 2D plane while minimizing the control energy (Flageat, Lim, Grillotti, et al., 2022). This environment is similar to the popular Ant-Uni environment (Chalumeau, Boige, Lim, et al., 2022) (as it involves the same articulated ant) but instead of trying to move as fast as possible

in a single direction, the goal is to reach any position on the surface. In this environment, the BD space is defined as the 2D plane and behavior descriptors are 2-dimensional vectors computed as the x and y positions of a solution at the end of an episode. We chose to restrict the BD space of Ant-Omni to $[-15, 15]$ on both axes as methods presented in this work tend to produce solutions within this range. The reward signal is defined as the negative energy consumption at each timestep and simply ensures that policies are constrained to produce energy-efficient behaviors. It is worth noting that this environment is primarily intended to evaluate the exploration abilities of algorithms, as the reward is relatively easy to optimize —an optimal policy would simply remain static. BD extraction function and fitness function for the Ant-Omni environment are defined as:

$$\begin{cases} \xi(\tau)_{\textit{Ant-Omni}} = (x_T, y_T) \\ F(\tau)_{\textit{Ant-Omni}} = -\sum_{t=1}^{T} ||\mathbf{a}_t||_2 \end{cases} \tag{4.2}$$

where T is the number of transitions in the trajectory, $||.||_2$ is the Euclidean norm, $\mathbf{a}_t$ is the action vector that corresponds to torques applied to the robot joints, and x_T and y_T the positions of the robot's center of gravity on both axes at the end of the trajectory.

Halfcheetah-Uni is a popular benchmark environment in which the agent must run as fast as possible in the forward direction while maximizing a trade-off between speed and energy consumption. In this environment, the BD space is defined as all the possible patterns of movement —or gaits—the bipedal agent can use to run. Behavior descriptors are defined as the proportion of time each foot of the agent is in contact with the ground. This definition is commonly used in other related works for tasks of similar nature (Cully, Clune, Tarapore, et al., 2015; Colas, Madhavan, Huizinga, et al., 2020; Pierrot, Richard, Beguir, et al., 2022). The reward signal is defined as the forward distance covered between each timestep minus a penalty for energy consumption. BD extraction function and fitness function for the Halfcheetah-Uni environment are defined as:

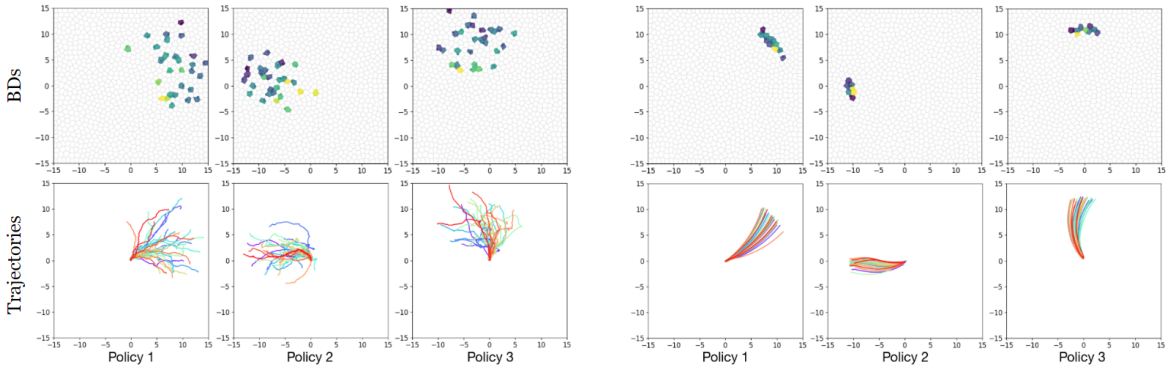
$$\begin{cases} \xi(\tau)_{\textit{Halfcheetah-Uni}} = (\frac{1}{T}\sum_{t=1}^{T} c_{i,t})_{i=1,2} \\ F(\tau)_{\textit{Halfcheetah-Uni}} = \sum_{t=1}^{T} \frac{x_t - x_{t-1}}{\Delta_t} - \sum_{t=1}^{T} ||\mathbf{a}_t||_2 \end{cases} \tag{4.3}$$

where x_t is the position of the robot's center of gravity at time t along the forward axis and Δ_t the timestep, and $c_{i,t} = 1$ if leg i is in contact with the ground at time t and 0 otherwise.

4.3.2 The Reproducibility Problem

Uncertain domains such as continuous control environments are known to be challenging for evolutionary methods. Apart from the fact that these methods are usually less efficient

when facing high-dimensional search spaces (Colas, Madhavan, Huizinga, et al., 2020), they also tend to generate policies that exhibit a high degree of variability in their behaviors and performances (Flageat and Cully, 2020; Flageat, Chalumeau, and Cully, 2022). As a result, repertoires of solutions generated by QD algorithms have limited re-usability in the sense that solutions they store rarely replicate the behaviors and performances for which they were retained. Even though the dynamics of the environment and the policy themselves may be deterministic, an ideal policy should be robust to varying initial conditions and demonstrate consistent behaviors and performances, particularly in the context of simulated robotics where the transfer to real applications is dependent on the robustness of such policies.

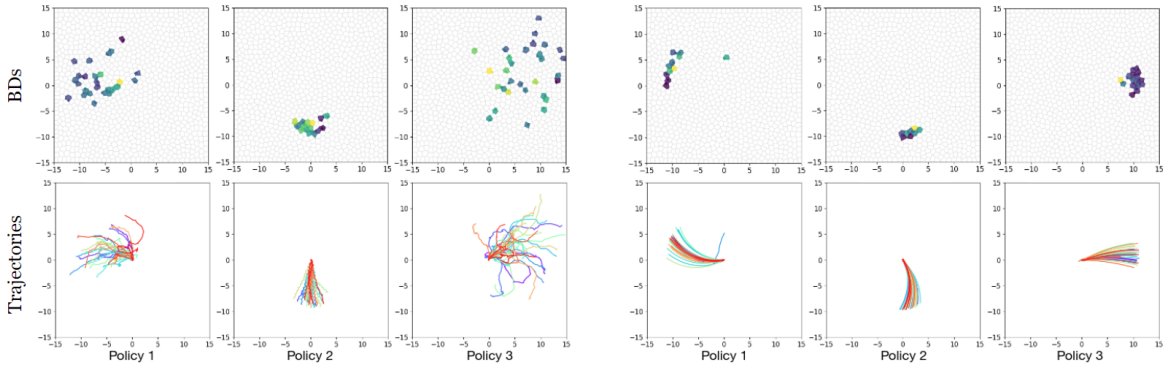


(a) MAP-Elites (b) MAP-Elites Low-Spread

Figure 4.3 – Reproducibility problem in Ant-Omni. We select three representative policies from final repertoires that have been generated by a) MAP-Elites and b) MAP-Elites Low-Spread, our proposed variant, and play 30 episodes with each policy using slightly varying initial states. The top row depicts the final BDs obtained by each policy and the bottom row represents the corresponding entire trajectories in the behavior space. Each color represents a different random seed (initial condition).

Figure 4.3 illustrates this problem for MAP-Elites in the Ant-Omni environment. It should be noted that we have made our best effort to select representative policies from the repertoire and not to cherry pick them. The reproducibility problem is actually twofold: first, it appears that the policies from the MAP-Elites repertoire exhibit a very high spread in the behavior space (as defined by Equation 4.1), meaning that a policy is unable to reproduce the results that were used to insert it in the repertoire. Table 4.4 contains average spread values for MAP-Elites and PGA-MAP-Elites in both environments. We argue that this pitfall of MAP-Elites based algorithms comes from the single evaluation scheme being used (see the MAP-Elites pseudocode in Algorithm 1) which drives the search process towards solutions that show high variability and that have been lucky during their single evaluation episode. Second, it appears that not only the MAP-Elites policies display a high spread in the behavior space, but they also produce inconsistent and irregular trajectories as depicted in the bottom row of Figure 4.3a. We refer to these trajectories as being irregular because the ant robot does not move steadily from

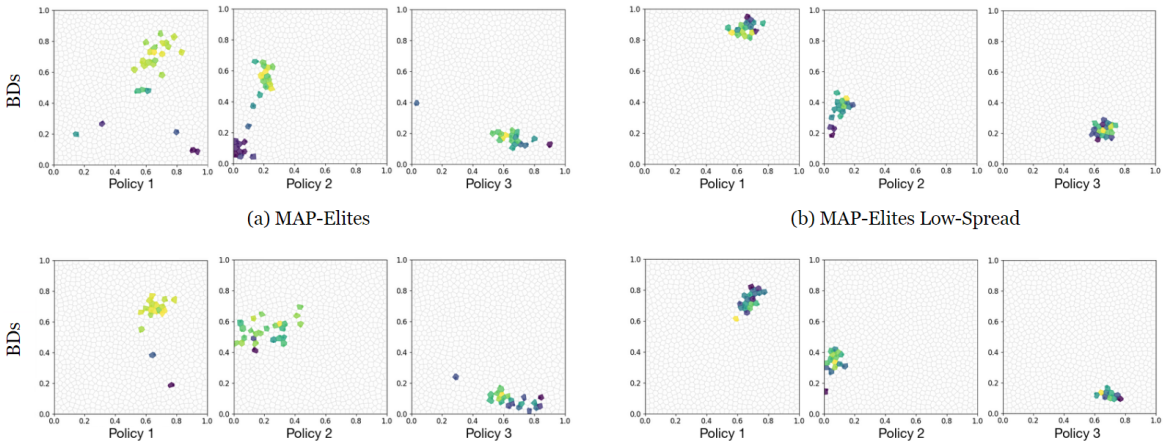
the starting point to its final position, but rather follows a shaky trajectory that often changes course.



(a) PGA-MAP-Elites (b) PGA-MAP-Elites Low-Spread

Figure 4.4 – Reproducibility problem in Ant-Omni. Here we reproduce the experiment proposed in Figure 4.3 for a) PGA-MAP-Elites and b) PGA-MAP-Elites Low-Spread.

Similar analyses hold for PGA-ME, Figure 4.4a illustrates the reproducibility problem in Ant-Omni for PGA-ME. We selected 3 representative policies from a final repertoire produced by PGA-ME and ran multiple ($N = 30$) episodes with each policy. Results are similar to ME policies in that they generate irregular trajectories and demonstrate high spread in the behavior space even though PGA-ME incorporates a policy-gradient-based mutation operator during its training process. Finally, we reproduce these experiments and show in Figure 4.5 that the same observations can be made in Halfcheetah-Uni for both algorithm families: ME and PGA-ME produce solutions that display high spread in the BD space. For this environment we do not show whole trajectories as the behavior space is of a different nature and not suitable for such plots.



(a) MAP-Elites (b) MAP-Elites Low-Spread

(c) PGA-MAP-Elites (d) PGA-MAP-Elites Low-Spread

Figure 4.5 – Reproducibility problem in Halfcheetah-Uni for a) MAP-Elites, b) MAP-Elites Low-Spread, c) PGA-MAP-Elites, d) PGA-MAP-Elites Low-Spread.

In the next section, we propose a new approach that augments MAP-Elites based algorithms with an additional insertion criterion based on the spread computation and show that it solves both of the above mentioned problems. We later demonstrate that the QDT alone is able to mitigate these problems but also that the supervised learning phase of the QDT largely benefits from trajectories that have been generated by steady and consistent policies.

4.4 Methods

In this section we first introduce MAP-Elites Low-Spread and PGA-MAP-Elites Low-Spread, which are modified versions of MAP-Elites (Mouret and Clune, 2015) and PGA-MAP-Elites (Nilsson and Cully, 2021), designed to solve the reproducibility problem introduced in Section 4.3.2. Then, we present the Quality-Diversity Transformer, a unique model compressing a repertoire of QD policies, capable of producing desired behaviors on demand thanks to policy distillation and supevised learning. All algorithms are implemented based on the QDax[1] framework (Chalumeau, Lim, Boige, et al., 2024), which provides highly parallelized versions of many state-of-the-art QD and RL algorithms using the Brax physics engine (Freeman, Frey, Raichuk, et al., 2021), and allows to create custom algorithms by providing basic building blocks. The duration of a MAP-Elites (resp. PGA-MAP-Elites) run requiring hundreds of millions of interactions with the environment does not exceed a few hours on modern GPUs using QDax.

4.4.1 MAP-Elites Low-Spread

MAP-Elites Low-Spread (ME-LS), our variant of the original MAP-Elites algorithm (ME), thrives the search process towards solutions that are consistent in the behavior space for uncertain domains. The full pseudocode is presented in Algorithm 6. ME-LS uses the global structure of ME except for two aspects. First, solutions are evaluated over multiple episodes ($N = 10$) and second, solutions are inserted into the repertoire if they prove to have a higher fitness *and* a lower spread than the solutions already contained. More precisely, the overall operation of ME-LS can be described in 3 principal steps: 1. ME-LS create new solutions through mutations, 2. It evaluates them over multiple episodes and computes their average fitnesses and their most frequent BDs[2], 3. These new solutions are inserted in the repertoire if they have better fitness and lower spread than the already stored solutions, or if the corresponding cells are empty.

[1] https://github.com/adaptive-intelligent-robotics/QDax

[2] Since the behavior space is continuous, we consider that two trajectories have the same behavior descriptor if they both belong to the same cell in the repertoire.

Algorithm 6 MAP-Elites Low-Spread

Given Max iteration number I, Number of initialization solutions G, Number of evaluations per solution E, MAP-Elites repertoire $\mathbb{M}$
$iteration\_number \leftarrow 0$
while $iteration\_number < I$ **do** ▷ Main loop
  if $iteration\_number < G$ **then** ▷ Initialize by generating G random solutions
    $\mathbf{x}' \leftarrow random\_solution()$
  else ▷ Sampling and mutation
    $\mathbf{x} \leftarrow random\_selection(\mathbb{M})$
    $\mathbf{x}' \leftarrow random\_genetic\_mutation(\mathbf{x})$
  Evaluate $\mathbf{x}'$ over E trajectories ▷ Evaluation
  Compute fitness of $\mathbf{x}'$ as its avg. fitness over E trajectories
  Compute BD of $\mathbf{x}'$ as its most frequent BD over E trajectories
  Compute the spread of $\mathbf{x}'$ in the BD space as given by Eq. 4.1
  if $\mathbf{x}'$ fitness and spread are better than the corresponding solution in $\mathbb{M}$ **then**
    Insert $\mathbf{x}'$ in $\mathbb{M}$
  $iteration\_number \leftarrow iteration\_number + 1$

As shown in Figure 4.3b, the policies generated by ME-LS exhibit highly consistent BDs over 30 episodes and regular, steady trajectories in the behavior space. It is clear that the additional insertion criterion forces solutions to be consistent in the behavior space, and we hypothesize that this additional constraint indirectly forces the selection process towards solutions that produce smooth, regular trajectories. Table 4.1 gives the hyperparameters for ME and ME-LS.

Hyperparameter	Value
Environment batch size	1000
Policy hidden layers size	$[256, 256]$
Iso sigma	0.005
Line sigma	0.05

Table 4.1 – Hyperparameters for MAP-Elites and MAP-Elites Low-Spread in both environments.

4.4.2 PGA-MAP-Elites Low-Spread

PGA-MAP-Elites Low-Spread (PGA-ME-LS) (see Algorithm 7) is analogous to ME-LS and simply modifies the PGA-ME algorithm to include an additional constraint over the policies spread during the insertion phase. Its overall structure is identical to the standard PGA-ME described in Section 4.2, except for the fact that PGA-ME-LS evaluates solutions over multiples trajectories and insert new solutions into the repertoire only if they present higher fitnesses *and* lower spreads than their corresponding solutions in the repertoire.

Conclusions similar to those of ME-LS can be drawn from Figure 4.4b for PGA-ME-LS, which does not suffer from the inconsistency problems of PGA-ME and trains policies that

Algorithm 7 PGA-MAP-Elites Low-Spread

Given Max iteration number I, Number of evaluations per solution E, Sample size N, MAP-Elites repertoire $\mathbb{M}$, Replay Buffer $\mathbb{B}$, A critic network Q_v
Initialization: Create N random policies $\{\pi_{\theta_i}\}_{i=1}^{N}$
Evaluate and insert them in $\mathbb{M}$
$iteration\_number \leftarrow 0$
while $iteration\_number < I$ **do** ▷ Main loop
    Sample N policies $\{\pi_{\theta_i}\}_{i=1,N}$ in repertoire $\mathbb{M}$ ▷ Sampling and mutation
    Mutate half the policies using the TD3 update using Q_v
    Mutate the other half with random genetic mutations
    ▷ Train the critic
    Sample batches of transitions in replay buffer $\mathbb{B}$
    Update the critic Q_v using TD3
    ▷ Evaluation
    Evaluate each new policy over E trajectories and store all transitions in buffer $\mathbb{B}$
    Compute each policy's fitness as its avg. fitness over the E trajectories
    Compute each policy's BD as its most frequent BD over the E trajectories
    ▷ Insertion in repertoire
    for each new policy **do**
        if fitness and spreadd of new policy are better than the corresponding policy in $\mathbb{M}$ **then**
            Insert new policy in $\mathbb{M}$
    $iteration\_number \leftarrow iteration\_number + 1$

Hyperparameter	**Value**
Environment batch size	1000
Policy learning rate	0.001
Critic learning rate	0.0003
Policy hidden layers size	$[256, 256]$
Critic hidden layers size	$[256, 256]$
Policy noise	0.2
Noise clip	0.5
Discount	0.99
Reward scaling	1.0
Policy gradient proportion	50%
Critic training steps	300
Policy training steps	100
Iso sigma	0.005
Line sigma	0.05

Table 4.2 – Hyperparameters for PGA-MAP-Elites and PGA-MAP-Elites Low-Spread in both environments.

are consistent in the BD space, producing smooth and regular trajectories. Table 4.2 provides hyperparameters for PGA-ME and PGA-ME-LS.

Full results comparing ME, ME-LS, PGA-ME and PGA-ME-LS are available in Section 4.5.4. As a quick note, in both environments, the average spread values presented in Table 4.4 indicate that simply performing multiple evaluations to better characterize a solution (as in ME-Sampling) does not help to increase its consistency in the behavior space. To solve this we argue that it is preferable to optimize directly for this purpose.

4.4.3 The Quality-Diversity Transformer

This section introduces the Quality-Diversity Transformer (QDT), a model inspired by the Decision Transformer (Chen, Lu, Rajeswaran, et al., 2021) that autoregressively models trajectories and produces behaviors on-demand by conditioning on a target behavior descriptor, as shown in Figure 4.6. Although very similar, the Quality-Diversity Transformer differs from the DT in that it conditions decisions on a target behavior rather than on the return to be obtained during an episode. Algorithm 8 contains the pseudocode for the QDT model, the supervised training loop and the evaluation loop.

A key aspect of the model is the trajectory representation used as input:

$$\tau = (\text{BD}, O_0, A_0, \text{BD}, O_1, A_1, ..., \text{BD}, O_T, A_T).$$

This trajectory structure enables the model to learn useful patterns and to conditionally generate actions at evaluation time. It should be noted that, contrary to the Decision Transformer, we simply use the same conditioning BD at each timestep and do not use a representation analogous to the return-to-go, which is a dynamic conditioning introduced in Chen, Lu, Rajeswaran, et al. (2021) to capture the return to be achieved until the end of the episode at any timestep. We tested it and obtained poorer results. We also tested a version of the QDT where the conditioning BD appears only once at the beginning of the episode, the intuition being that the attention mechanism of the Transformer should be able to focus on this element, even if it appears only once. Results were very similar to those presented in this chapter but marginally inferior, hence the choice of preferring this representation. We feed trajectories of length $3T$ to the QDT, as we have 3 elements at each timestep (one for each modality: conditioning BD, observation and action). Note that the QDT takes raw continuous inputs from the environment which are not normalized. We compute embeddings for these elements by learning a linear layer for each modality, which projects elements to the embedding dimension, followed by layer normalization (Ba, Kiros, and Hinton, 2016). Timestep embeddings are learned for each of the T timesteps and added to their corresponding elements. This differs from traditional positional embeddings used in Transformers as one timestep embedding corresponds to three

elements. Finally, the trajectory is processed by the Transformer model which predicts one continuous action vector for each timestep.

Algorithm 8 Quality-Diversity Transformer

Given:
- Target BD, Observations, Actions: BD, O, A
- Causal Transformer model (GPT) $Transformer$
- Embedding layers for each modality: E_{BD}, E_O, E_A
- Timestep embedding layer E_t
- Linear action prediction layer $Pred_A$
- Episode length T

// QDT model
function $QDT(\text{BD}, O, A, t)$
    timestep_emb = $E_t(t)$ ▷ Compute inputs embeddings
    BD_emb = $E_{\text{BD}}(\text{BD})$+ timestep_emb
    O_emb = $E_O(O)$+ timestep_emb
    A_emb = $E_A(A)$+ timestep_emb
    // Interleave inputs as $(BD, O_1, A_1, ..., BD, O_T)$
    inputs_emb = interleave(BD_emb, O_emb, A_emb)
    hidden_states = $Transformer$(inputs_emb) ▷ Process inputs with transformer
    return $Pred_A$(hidden_states) ▷ Predict actions

// Training loop
// Dims: (batch_size, T, dim)
for (BD, O, A, t) **in** dataloader **do**
    A_preds = $QDT(\text{BD}, O, A, t)$
    loss = mean($(A\_preds - A)^2$)
    optimizer.zero_grad(); loss.backward(); optimizer.step()

// Evaluation loop (autoregressive generation)
BD = generate_target_BD()
BD, O, A, t, done = [BD], [env.reset()], [], [1], False
while not done **do**
    action = $QDT(\text{BD}, O, A, t)[t]$ ▷ Sample next action
    new_O, done = env.step(action)
    BD = BD + [BD] ▷ Append new elements to sequence
    $O, A, t = O + [new\_O], A + [action], t + [len(\text{BD})]$

During training, we use a dataset of offline trajectories generated by policies from a QD repertoire and leverage supervised learning to train the QDT over entire trajectories from the dataset, which is a point of divergence with the Decision Transformer as they use randomized reduced windows instead of whole trajectories. The prediction head corresponding to the input token O_t is trained to predict A_t with mean-squarred error loss as we run our experiments on continuous action spaces (see Figure 4.1 part 3: "Supervised Learning"). Because of the trajectory structure and since the model is a causal Transformer (GPT-2 based, Radford, Wu, Child, et al., 2019), we make forward passes over minibatches of entire trajectories to predict actions for all timesteps at once and compute the global loss over all timesteps and all trajectories

of the minibatch[3]. Still, the QDT is not allowed to cheat during training because its causal nature implies that it can only attend to previous inputs at any given timestep; as depicted in Equation 2.2, the GPT architecture implements a causal attention mechanism masking all future positions to ensure causality. Table 4.3 summarizes QDT hyperparameters for all experiments.

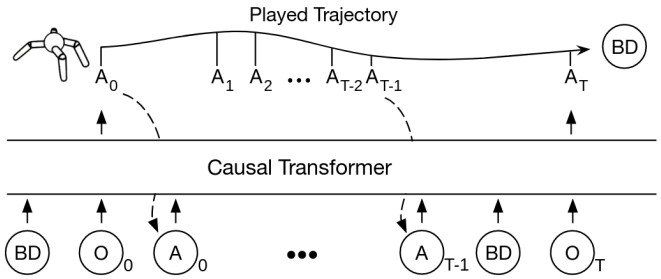


Figure 4.6 – The QDT autoregressively plays an evaluation episode by conditioning on a target BD. At any given timestep t, the QDT generates an action by looking at all elements of the trajectory that precede t. The first action A_0 is generated given the target BD and the first observation O_0, while the last action A_T is generated given the whole trajectory of target BDs, observations and previously generated actions. Note that the target BD is the same for the entire trajectory.

To evaluate the QDT, we simply condition it on a target behavior descriptor (BD) and feed it with the first observation as given by the environment. The model generates the first action, which is played next in the environment and appended to the trajectory of inputs for the next inference. We unroll a whole episode in the environment following this procedure and measure the QDT performance by computing the Euclidean distance between the conditioning BD and the BD actually achieved by the model during the episode. Figure 4.6 depicts an evaluation episode played by the QDT. In practice, to speed up experiment durations, we parallelize the evaluation phase over multiple goals at the same time, and conduct these parallelized evaluations on distributed devices (with varying initial seeds) to obtain averaged metrics for all goals at once.

Hyperparameter	**Value**
Number of layers	4
Number of attention heads	8
Embedding dimension	256
Nonlinearity function	ReLU
Batch size	256
Dropout	0.1
Learning rate	0.0007

Table 4.3 – Hyperparameters for the Quality-Diversity Transformer in both environments.

[3]This training procedure is logically much faster than using batches of randomly truncated trajectories and predicting a unique action per trajectory (for the last timestep only).

4.4.4 Dataset Creation

Dataset generation is the second stage of our full pipeline illustrated in Figure 4.1. For this purpose, we divide the repertoire into large zones (50 for Halfcheetah-Uni, 100 for Ant-Omni) and select the best policy for each zone. To do so, each candidate policy of a given zone plays a few evaluation episodes and the policy that most often produces BDs corresponding to the zone is selected. We thus obtain a total of 50 (resp. 100) selected policies, and make them play trajectories that are stored into a dataset. Datasets for both environments are constituted of $300,000$ trajectories, or equivalently 75 million transitions, given that each trajectory consists of 250 time steps.

4.5 Results

In this section, we present results for all methods introduced in Section 4.4. First we propose an accuracy experiment to compare the ability of different methods to accurately produce a BD on demand, then we assess the generalization capabilities of the QDT. Finally we provide detailed comparisons of the QD algorithms (ME, PGA-ME, ME-LS and PGA-ME-LS).

4.5.1 QDT Training and Ablations

During the supervised learning process, we periodically evaluate the QDT and report its average Euclidean distance to target BDs. For each evaluation phase, these target BDs (or goals) are chosen to be representative of the whole behavior space, as we want to assess the model capacity to reach all zones of the space with high accuracy. To do so, we compute a centroidal Voronoi tesselation of the BD space and use each centroid as a conditioning goal for the QDT. We use 50 and 100 evaluation goals for Halfcheetah-Uni and Ant-Omni respectively, although these values are arbitrary, we consider that they allow a fair coverage of the BD space in both cases. We run multiple ($N = 10$) episodes for each goal and compute the average distance per goal —which can be visualized as shown in Figure 4.8— as well as the overall average distance for all goals.

Figure 4.7 plots the overall average distance in both environments for different models through the training process. In both tasks, we train the models for 100 epochs and perform evaluation every 5 epochs. Models are named after the QD algorithm that was used to generate the dataset: QDT(ME-LS) refers to the QDT that is trained on a dataset generated by MAP-Elites Low-Spread policies, which constitutes our full algorithm as depicted in Figure 4.1. QDT(ME) is an ablation refering to the QDT that is trained on a dataset generated by MAP-Elites policies, and so on for PGA-ME and PGA-ME-LS. The QDT(Naive) model corresponds to an ablation

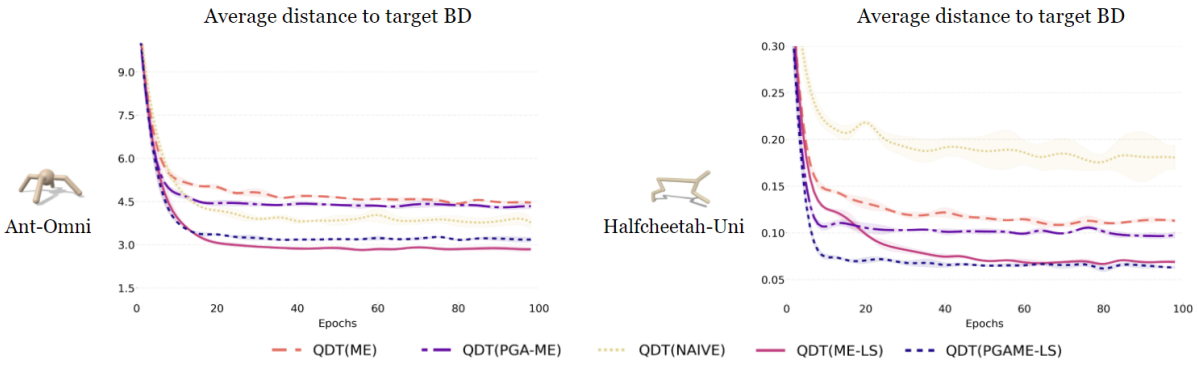


Figure 4.7 – Evaluation results of the QDT through the training process (average values and std ranges on 3 seeds). We evaluate the model over multiple goals (target BDs) covering the behavior space and report the total average Euclidian distance to these goals. The models trained on datasets created by Low-Spread methods, whether using ME-LS or PGA-ME-LS, show significantly better performances.

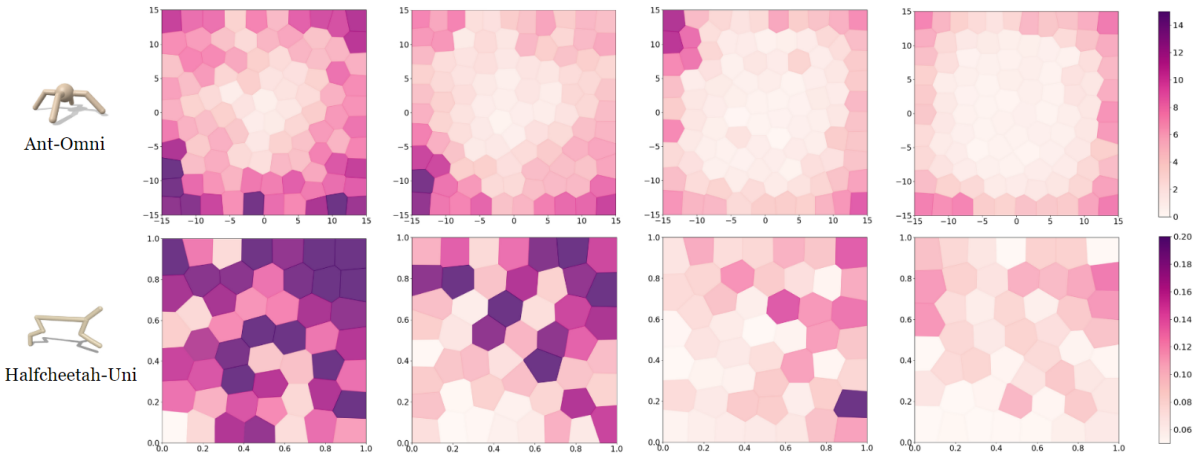
where we do not apply the dataset creation method described in Section 4.4.4 and simply run all policies from a ME-LS repertoire to generate the dataset.

Results show that the trend is the same among the two algorithm families (ME and PGA-ME): Low-Spread based QDTs outperform their vanilla counterparts and achieve significantly lower average distance to target BDs. We argue that this is the direct result of using more consistent policies to create the dataset, and we hypothesize that learning to replicate a skill that has been demonstrated in a steady and accurate manner is inherently easier than learning to replicate irregular demonstrations, as depicted in Figure 4.4. Lastly, it appears that the dataset creation method of Section 4.4.4 is crucial to the good performance of the model, whose QDT(Naive) ablation especially struggles in Halfcheetah-Uni.

4.5.2 Accuracy Experiment

This experiment aims to answer the following question: **Which method is more accurate and consistent in achieving a target behavior?** To answer this question, we test the ability of ME, ME-LS, QDT(ME) and QDT(ME-LS) to achieve target BDs on demand and measure their consistency and accuracy. Similar to the evaluation method described in Section 4.5.1, we choose evaluation goals that cover the behavior space and run multiple evaluation episodes ($N = 10$) for each goal. For ME and ME-LS, we run the solution of the repertoire which is closest to each respective goal. For QDT models, we select them according to their best training epoch (see Figure 4.7) and condition them directly on the goals (target BDs).

Figure 4.8 depicts the average distance per goal for each method. Color indicates whether, on average over the 10 episodes, we were able to reach the aimed point in the BD space (lighter is better). Importantly, it should be noted that in Ant-Omni, the robot has a size of roughly 2

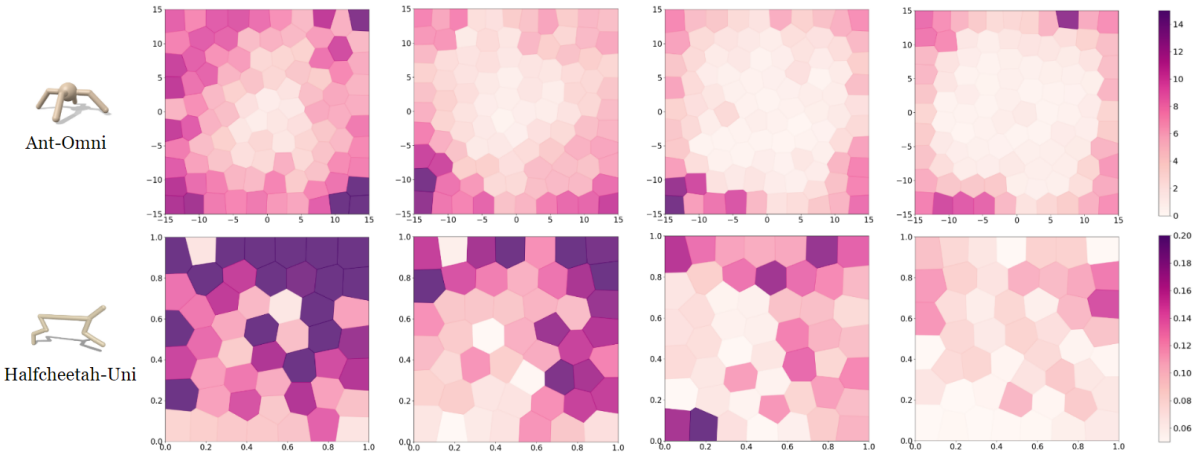


(a) ME (b) QDT(ME) (c) ME-LS (d) QDT(ME-LS)

Figure 4.8 – Results of the accuracy experiment. This experiment can be described in 2 steps: 1. We select multiple evaluation goals (target BDs) in the behavior space, 100 and 50 for Ant-Omni and Halfcheetah-Uni respectively. To get meaningful goals, we simply compute a CVT of the BD space in which goals are the centers of each zone, 2. For each goal, we play 10 episodes and plot their average Euclidean distance to the goal. For ME and ME-LS, trajectories are played by the nearest policy to the goal in the repertoire. For the QDT, we simply condition it on the goal. Distance is represented by color: lighter is better. The QDT(ME-LS) appears to be the most accurate method to achieve behaviors on demand.

units of distance and navigates on a plane of 30 units of distance on both axes. We consider achieving a distance to the target BD ≤ 2 a good performance. Likewise, the BD space of Halfcheetah-Uni ranges from 0 to 1 on both axes and we consider a distance to the target BD ≤ 0.1 a good performance. In both environments, results show that ME solutions hardly achieve the targeted behaviors. The QDT(ME) significantly improves over ME policies but is still short of producing accurate BDs. We hypothesize that the Transformer model helps to generalize on the data produced by ME policies but is still hampered by their irregular trajectories (we discuss the generalization abilities of the QDT in the next section). Finally, ME-LS and QDT(ME-LS) both demonstrate high accuracy for almost all goals in both environments.

We show that similar results and conclusions hold for PGA-ME and its variations in Figure 4.9, which depicts the exact same accuracy experiment. Similarly to results presented in Figure 4.8, PGA-ME largely fails to achieve target BDs on demand, while the QDT(PGA-ME) improves over this result. The QDT(PGA-ME-LS) appears to be the most accurate method to achieve target BDs in both environments, which is consistent with results obtained for the ME family where the QDT(ME-LS) obtained best results. Importantly, note that all methods struggle to reach the most outer goals in Ant-Omni, this is due to the fact that no policy –hence no data– is available for these zones of the BD space as shown in the dataset representation in upper part of Figure 4.10.



(a) PGA-ME (b) QDT(PGA-ME) (c) PGA-ME-LS (d) QDT(PGA-ME-LS)

Figure 4.9 – Results of the accuracy experiment for PGA-ME and its variants. The experimental process is strictly identical to that of Figure 4.8. Analogously to results of the MAP-Elites variants in Figure 4.8, QDT(PGA-ME-LS) appears to be the most accurate method to achieve behaviors on demand.

Table 4.4 repeats the accuracy experiment over multiple seeds and presents the overall average distance to goals and the overall average spread for all methods and for both environments. In line with the results reported in Figure 4.8, QDTs trained on datasets created by ME-LS and PGA-ME-LS policies show the best ability to achieve target behaviors while QDT(ME) and QDT(PGA-ME) confirm their superiority over ME and PGA-ME respectively. Additionally, we observe that the ME-Sampling baseline significantly improves over ME but is still far from the performance of ME-LS, particularly in Halfcheetah-Uni where we hypothesize that the geometric mean BD computation qualifies solutions in BD cells that they never actually achieve. While the Low-Spread versions clearly outperform their original counterparts in all comparisons, there is no clear winner between ME and PGA-ME families, neither in their raw use, nor in the use of their repertoire to train the QDT; PGA-ME-LS and QDT(PGA-ME-LS) does not clearly outperform ME-LS and QDT(ME-LS), respectively. This suggests that, at least for the environments and metrics considered, the use of a policy gradient mutation operator — present in PGA-ME —, does not help to produce more consistent solutions in the BD space.

Importantly, note that QDT models benefit from filtered policies that have been selected for their consistency as described in Section 4.4.4. As illustrated in Figure 4.7, this step is crucial and allows the model to train on a dense, consistent dataset for each zone of the BD space. In other words, it appears that the QDT benefits highly from supervised demonstrations where the data distribution comes from a reduced set of consistent policies (each of which showing how to achieve a different and distinct goal), rather than numerous policies overlapping in the BD space. Overall, during the experimentation process, all the tests undertaken to improve dataset quality proved to have great impact on the final accuracy of the QDT, as opposed to the

Table 4.4 – Results of the accuracy experiment (described in Figure 4.8). For each environment and algorithm, we present the average distance over all goals (target BDs) and the overall average spread. Each experiment is repeated over 5 seeds and reported with average values and standard deviations.

Method	Ant-Omni		Halfcheetah-Uni	
	Avg. dist.	Spread	Avg. dist. ($\times 10^2$)	Spread ($\times 10^2$)
ME	$6.38 \pm .15$	$3.76 \pm .03$	$16.33 \pm .47$	$15.45 \pm .45$
PGA-ME	$6.39 \pm .14$	$3.99 \pm .09$	$17.67 \pm .48$	$16.00 \pm .81$
ME-LS	$3.01 \pm .05$	$2.33 \pm .01$	$10.01 \pm .47$	$10.01 \pm .45$
PGA-ME-LS	$3.13 \pm .04$	$2.24 \pm .04$	$10.00 \pm .00$	$10.67 \pm .50$
ME-Sampling	$5.10 \pm .19$	$4.50 \pm .11$	$15.40 \pm .90$	$14.12 \pm .95$
QDT (ME)	$4.64 \pm .10$	$4.66 \pm .07$	$12.00 \pm .00$	$13.00 \pm .00$
QDT (PGA-ME)	$4.45 \pm .05$	$4.18 \pm .16$	$12.00 \pm .00$	$12.00 \pm .00$
QDT (ME-LS)	$\mathbf{2.86 \pm .07}$	$\mathbf{1.99 \pm .09}$	$9.09 \pm .32$	$9.20 \pm .22$
QDT (PGA-ME-LS)	$3.06 \pm .03$	$2.32 \pm .06$	$\mathbf{7.34 \pm .42}$	$\mathbf{8.40 \pm .37}$

changes made to the model architecture —and model hyperparameters—, which proved to have much less impact.

4.5.3 Generalization Experiment

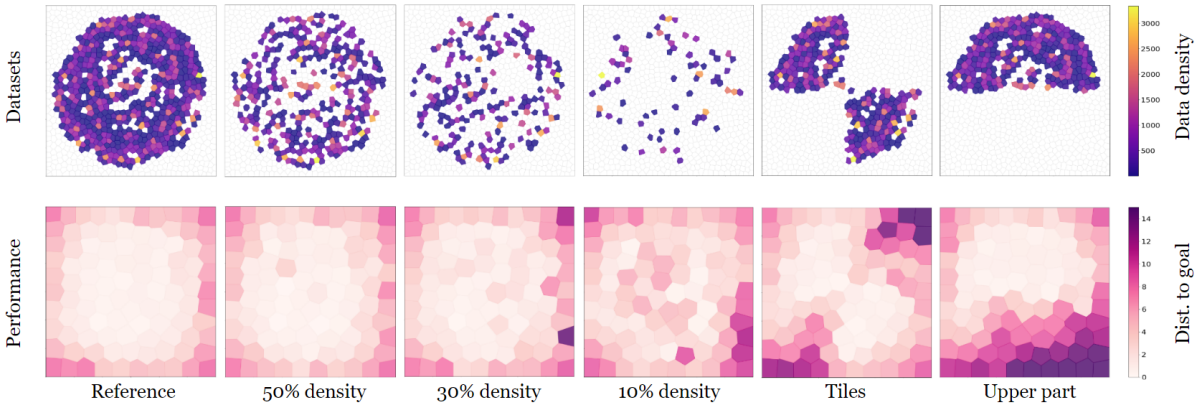


Figure 4.10 – Results of the QDT generalization experiment in Ant-Omni. In this experiment we run accuracy experiments (bottom row) on truncated datasets (top row) which are deprived of a part of their trajectories. The QDT shows strong interpolation ability on the 50%, 30% and 10% density datasets and a more limited ability to extrapolate in "Tiles" and "Upper part" datasets where entire zones of the BD space are deprived of data.

This section aims to understand the generalization abilities of the QDT. Importantly, we want to distinguish between interpolation, the model's ability to generalize between BDs existing in the dataset, and extrapolation, the model's ability to reach BDs that are beyond existing

examples. To do so, we sparsify datasets generated by ME-LS policies by pruning trajectories using different pruning schemes and train QDTs on these truncated datasets.

Figure 4.10 presents accuracy results for the Ant-Omni task. The top row shows the different datasets used in this experiment, color depicts the trajectory density in the BD space, light color corresponds to a high density zone. The bottom row shows accuracy experiments (similar to Figure 4.8) for the QDTs trained on the corresponding datasets. The 50%, 30% and 10% density datasets aim to measure the interpolation capacity of the QDT, while the "Tiles" and "Upper part" datasets aim to measure its extrapolation capacity. Our model demonstrates strong interpolation capabilities and is able to achieve behaviors with reasonable accuracy even in the 30% and 10% density settings. However, its extrapolation capabilities appear more limited as can be seen in the "Tiles" and "Upper part" settings.

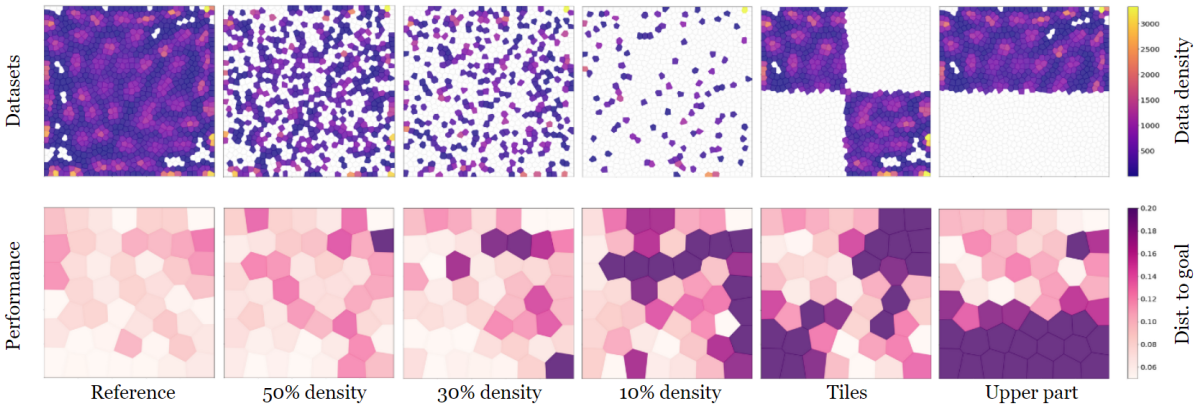


Figure 4.11 – Results of the QDT generalization experiment in Halfcheetah-Uni. The QDT shows strong interpolation ability on the 50%, and 30% density datasets and a limited ability to extrapolate in "Tiles" and "Upper part" datasets where entire zones of the BD space are deprived of data.

Figure 4.11 presents the generalization experiment for the Halfcheetah-Uni environment. We observe that, even though the QDT demonstrates good accuracy up to the 30% density setting, it has more difficulties to generalize in this environment, both for interpolation and extrapolation. We hypothesize that this difference between Ant-Omni and Halfcheetah-Uni comes from the very different nature of their behavior spaces. After the execution of a QD algorithm in Ant-Omni, two policies that are close in the BD space often produce similar full-body trajectories, meaning that they walk on the 2D plane and reach their final positions. Although these positions happens to be slightly different, both policies usually walk with similar gaits. In Halfcheetah-Uni, two policies that are close in the BD space can demonstrate radically different full-body behaviors. As an extreme example, it occurred that we observed neighboring policies in the BD space, one of which was doing backflips while the other was running normally. We believe that these gaps in real behaviors prevent effective generalization for the QDT. Finally, note that in these generalization experiments we simply prune trajectories

from the datasets and do not increase the number of trajectories in preserved zones, which can affect the model in the sense that it has strictly less data to train on.

4.5.4 QD Algorithms Results

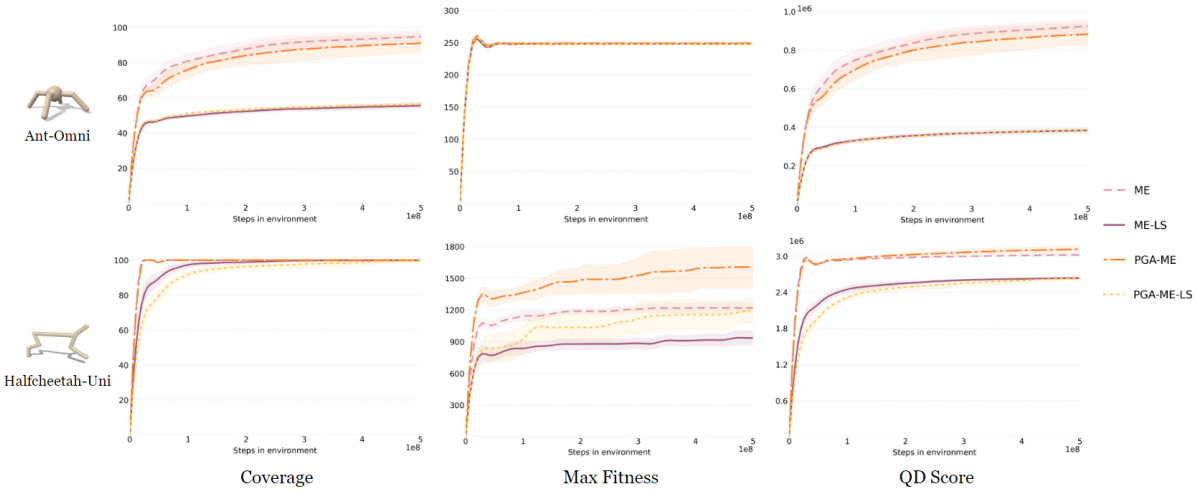
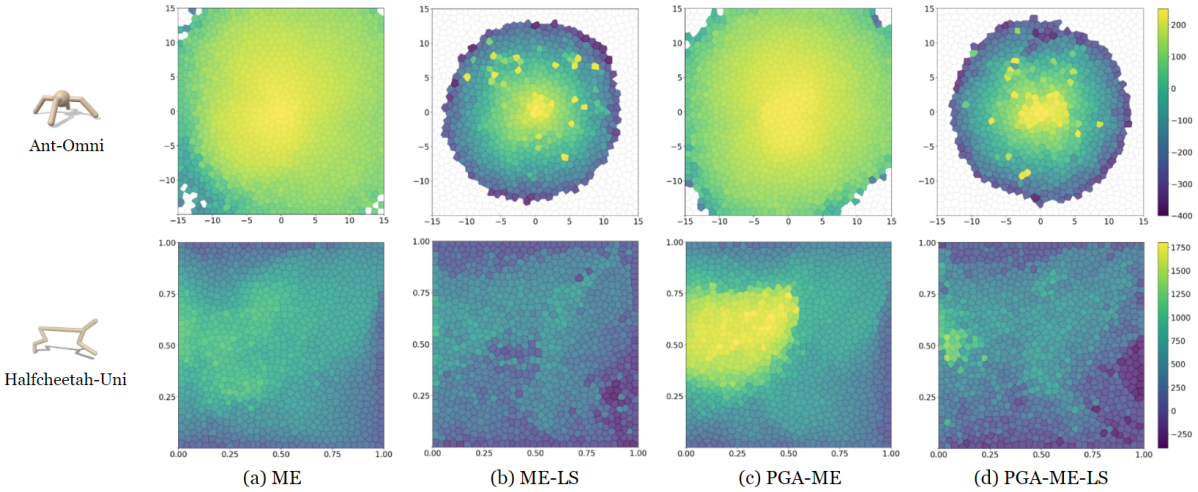


Figure 4.12 – Results of the Quality-Diversity algorithms: MAP-Elites (ME), PGA-MAP-Elites (PGA-ME) and their Low-Spread variations (ME-LS and PGA-ME-LS) in both environments over 5 seeds. Coverage indicates the proportion of the behavior space that have been covered in the repertoire, max fitness reports the best fitness obtained by any solution evaluated so far and the QD score represents the total sum of fitness across all solutions in the repertoire. Performances are plotted against the number of interactions with the environment.

Figure 4.12 and Figure 4.13 gather results for the QD algorithms runs, namely MAP-Elites (ME), PGA-MAP-Elites (PGAME) and their Low-Spread versions (ME-LS and PGAME-LS), and show their respective coverages, maximum fitnesses, and QD scores in both environments. In accordance with standard practices in QD research, we add an offset to the fitnesses when computing the QD score to ensure that it is an increasing function of the coverage. The initial repertoires, which are identical for all methods, are of size 1024 and are generated using Centroidal Voronoi Tessellations (Vassiliades, Chatzilygeroudis, and Mouret, 2016).

Results show that the original versions of these algorithms (ME and PGAME) obtain the best performances over all training metrics by a significant margin. However, it is important to recall that first, the original versions evaluate each policy only once, contrary to the Low-Spread versions that do multiple evaluations (usually 10), which strongly promotes accidental policies that have been lucky over their unique evaluation episode obtaining abnormally high fitnesses and inaccurate BDs. And second, The Low-Spread versions include an additional insertion criteria that drives the search process towards policies that are more consistent in the BD space, leading to fewer insertions in the repertoire —policies have to show higher average fitness *and* better consistency than their counterpart in the repertoire to be selected. We argue that these

very points are responsible for the difference in coverage and, *in fine*, in maximum fitness and QD score. But more importantly, although the original versions present better training metrics, they remain limited as these metrics do not take into account the actual usefulness of a QD repertoire and its capacity to deliver accurate and consistent solutions that behave according to their BDs, as shown in Section 4.3.2 and Figure 4.8. Finally, as shown in Figure 4.12, even though the Low-Spread versions require several evaluations per solution, their convergence rate (in terms of number of interactions with the environment) is in fact similar to that of original methods, resulting in a comparable sample efficiency.



(a) ME (b) ME-LS (c) PGA-ME (d) PGA-ME-LS

Figure 4.13 – Coverage maps of QD algorithms. Fitness is represented by color; lighter is better.

To determine the actual capabilities of QD repertoires produced by ME, PGA-ME and their Low-Spread counterparts, we propose a reassessment experiment on Halfcheetah-Uni where all stored solutions are evaluated again and multiple times. For each algorithm, we take a final repertoire of policies and test them again over multiple episodes. We insert them into a new, empty repertoire and report its coverage, max fitness and QD score. Table 4.5 gives the

Table 4.5 – Results of the reassessment experiment in Halfcheetah-Uni. "Initial" columns show values for the initial repertoire, that is, the repertoire that was used during the algorithm run. "Recalc." columns refer to values of the new repertoire that contains solutions after re-evaluation.

	Coverage (in %)		Max Fitness		QD Score ($\times 10^6$)	
	Initial	**Recalc.**	Initial	**Recalc.**	Initial	**Recalc.**
ME	100	**43**	1226	**770**	3.05	**1.17**
ME-LS	100	**55**	992	**730**	2.63	**1.41**
PGAME	100	**40**	1417	**977**	3.07	**1.10**
PGAME-LS	100	**53**	1194	**1073**	2.63	**1.36**

results of the reassessment experiment and show that the performance gap in training metrics

is not representative of the true quality of final repertoires. After re-evaluation, ME-LS and PGA-ME-LS repertoire obtain better coverages and QD Scores, and comparable maximum fitnesses to ME and PGA-ME.

4.5.5 QDT Fitness Results

Figure 4.14 reports the performances of all variants of the QDT in terms of fitness for Halfcheetah-Uni. During evaluation phases of the training process (described in Section 4.5.1), we record the average fitness obtained for each goal (target BD). The maximum fitness reported in Figure 4.14 simply corresponds to maximum over all goals. To be fair, these results should be compared to results of the reassessment experiment in Table 4.5 as we want to know what is the maximum average fitness that we can expect from each method at evaluation time. It appears that the QDT is able to reproduce the maximum fitnesses of the QD policies that were used to generate its dataset.

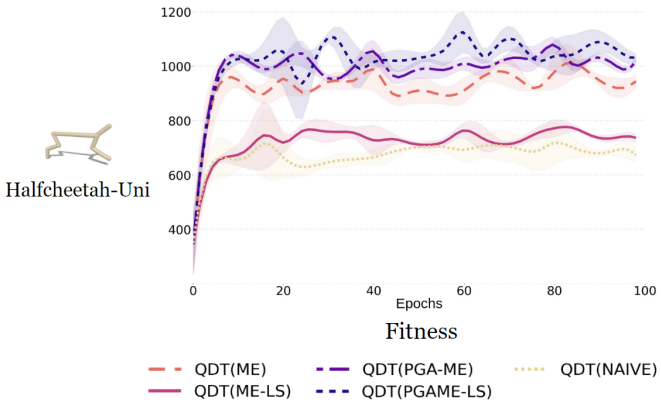


Figure 4.14 – Maximum fitness of the QDT in Halfcheetah-Uni for evaluations during the training phase (average values and std ranges on 3 seeds). We report the maximum fitness obtained over all goals. Performances are similar, if not superior, to values reported in the reassessment experiment in Table 4.5, meaning our model is able to replicate the fitness of QD policies.

4.6 Conclusion

In this chapter, we presented MAP-Elites Low-Spread, a QD algorithm that allows the neuroevolution of diverse and consistent solutions in uncertain domains, and the Quality-Diversity Transformer, a single policy achieving target behaviors with high accuracy by using behavior-conditioning. We showed that the QDT benefits from steady trajectories generated by ME-LS policies and that it is the best option to achieve behaviors on demand while being able, to some extent, to generalize to unseen zones of the BD space. We believe that an interesting future

direction would be to apply the QDT to non-uncertain domains where there is no need to couple it with a Low-Spread-based QD algorithm, and benefit from its generalization capacities.

Acknowledgements

The work in this chapter was supported by Cloud TPUs from Google's TPU Research Cloud (TRC) and has received funding from the European Commission's Horizon Europe Framework Program under grant agreement No 101070381 (PILLAR-robots project).

Chapter 5

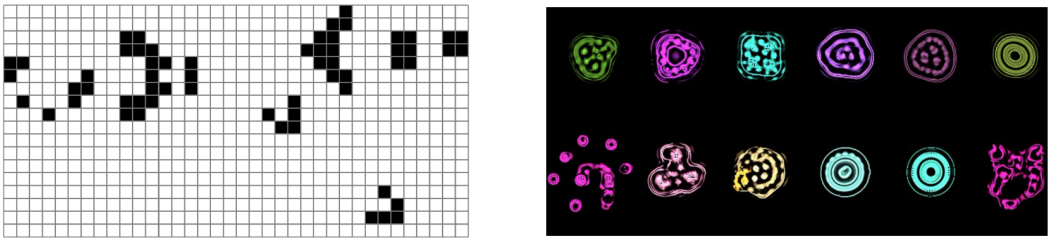
Harnessing Deep Learning for Diversity Search in Continuous Cellular Automata

I was born not knowing and have had only a little time to change that here and there.

Richard Feynman.

This chapter presents a work in progress in which we propose to train a Transformer-CNN model to imitate continuous cellular automata, and predict patterns generation solely from the initial state. We argue that leveraging this model in a divergent search algorithm could help automate the search for unique and interesting patterns.

Contents



(a) (b)

Figure 5.1 – Cellular automata are mathematical systems enabling the emergence of complex structures from simple rules. They can be discrete, as in the Game of Life (a) (Gardner, 1970), or continuous, as in Flow Lenia (b) (Plantec, Hamon, Etcheverry, et al., 2023).

5.1 Introduction

Cellular automata (CA) (Wolfram and Gad-el-Hak, 2003) have long been a subject of interest in the study of complex systems, with John Conway's *Game of Life* (Gardner, 1970; Berlekamp, Conway, and Guy, 2004) being the most famous example. A cellular automaton usually consists of a regular grid of cells, each of which can be in a set of defined states. The initial grid configuration evolves over discrete timesteps according to simple rules based on the states of neighboring cells, i.e., the state of a given cell at timestep $t + 1$ can be computed using the set of rules governing the CA, and with respect to neighbor states at timestep t. Apart from being a mathematical curiosity, cellular automata were initially developed by John Conway with the intuition that complex systems can emerge through simple rules, his motivation was to create a mathematical model that could simulate the way complex patterns and structures evolve in nature. Figure 5.1a provides an illustration of a distinctive pattern in the Game of Life, the *Gosper glider gun*, which repeatedly emits the well-known *glider* patterns. Interestingly enough, and as a side note, the original Game of Life has been proven Turing complete (Rendell, 2002), and the Gosper glider gun constitutes the basic building block for all logical gates.

> **Remark 5.1** (Emergence and intelligent design)**.** *Cellular automata provide an early intuition that simple rules can lead to complex structures, and we can see it unfold before our eyes. Learning cellular automata brings the benefit of augmenting one's resilience to intelligent design arguments, which are common when it comes to explaining new phenomena.*

Beyond theoretical computer science, cellular automata have been used in modeling natural phenomena. For instance, they have been employed to simulate the spread of fires in forest ecosystems, helping researchers understand and predict wildfire dynamics (Drossel and Schwabl, 1992). In the study of fluid dynamics, lattice gas automata have been utilized to

model fluid flow at microscopic scales (Frisch, Hasslacher, and Pomeau, 1986). Additionally, cellular automata have found applications in biological modeling (Ermentrout and Edelstein-Keshet, 1993).

Building on the classical framework of cellular automata, continuous cellular automata (Rafler, 2011; Chan, 2018; Mordvintsev, Niklasson, and Randazzo, 2023) extend the concept to continuous space, time and states. This shift allows for the simulation of more complex phenomena that are not easily captured by discrete models. Figure 5.1b provides an image of a simulation in the continuous CA Flow Lenia (Plantec, Hamon, Etcheverry, et al., 2023), where different patterns emerge, reminiscent of primitive life forms.

In this chapter, we propose a new method to search for interesting patterns in cellular automata. We train an hybrid Transformer-CNN model (Krizhevsky, Sutskever, and Hinton, 2012; Vaswani, Shazeer, Parmar, et al., 2017) to predict the evolution of a CA simulation from the initial CA configuration at timestep $t = 0$ without knowing the rules that govern it, and we propose to use the error of the trained model to drive the search process of an evolutionary algorithm towards novelty.

5.2 Background

Although the method introduced in this chapter is intended to be generic and not restricted to a specific cellular automata, we have opted to use Flow Lenia (Plantec, Hamon, Etcheverry, et al., 2023) as a support for our experiments, mainly for its ability to generate interesting and stable patterns due to its main specificity: the mass conservation. In this section, we present the theoretical background of Flow Lenia, starting with the original Lenia (Chan, 2018).

For the following, let $\mathcal{L}$ be the two-dimensional grid $\mathbb{Z}^2$ supporting the CA. Let $A^t : \mathcal{L} \to \mathcal{S}^C$ be the activations at timestep t, where $\mathcal{S}$ is the state space and C the number of channels of the system. Contrary to the Game of Life, which has only one channel, more complex cellular automata often feature multiple channels, potentially interacting with one another. Thus, $A_i^t(x)$ represents the activation in $x \in \mathcal{L}$ for channel i at timestep t. Put simply, in discrete CAs, such as the Game of Life, the activation is the binary state associated to a cell (*dead* or *alive*), while in continuous CAs, this state can take real values.

Lenia: In the original Lenia, which is the base CA Flow Lenia builds on, the state space $\mathcal{S}$ spans the continuous range $[0, 1]$. An instance of Lenia is characterized by a tuple (K, G, A^0), where A^0 is the initial configuration of activations in the grid, and K is a collection of convolution kernels such that each kernel K_i satisfies $\int_{\mathcal{L}} K_i = 1$. Kernels are designed with radial symmetry

and consist of concentric Gaussian bumps, defined as:

$$K_i(x) = \sum_{j=1}^{k} b_{i,j} \exp\left(-\frac{\left(\frac{x}{r_i R} - a_{i,j}\right)^2}{2w_{i,j}^2} \right),$$

where a_i, b_i, w_i and r_i are parameters defining kernel i, k is a parameter defining the number of rings per kernel, and R is a parameter common to all kernels defining the maximum neighborhood radius. Each kernel is then defined by $3 \times k + 1$ parameters.

G is a set of growth functions such that $G_i : [0,1] \to [-1,1]$. Intuitively, these growth functions act as the rules of the CA that govern the evolution of patterns during the simulation for a given location x (see Equation 5.1). In Lenia, growth functions are Gaussian functions scaled to $[-1,1]$ defined as:

$$G_i(x) = 2\exp\left(-\frac{(\mu_i - x)^2}{2\sigma_i^2} \right) - 1, \tag{5.1}$$

where μ_i and σ_i are the two parameters of the growth function i. Each pair (K_i, G_i) is connected to a source channel c_0^i it senses and a target channel c_1^i it influences. This connectivity is expressed via a square adjacency matrix $M_{C,C}$ as shown:

$$M_{C,C} = \begin{bmatrix} m_{11} & \cdots & m_{1C} \\ \vdots & \ddots & \vdots \\ m_{C1} & \cdots & m_{CC} \end{bmatrix}$$

where $m_{ij} \in \mathbb{N}$ denotes the number of kernels that sense channel i and influence channel j. A simulation timestep in Lenia consists of two substeps:

1. Compute the growth U_j^t at timestep t for channel j given A^t:

$$U_j^t = \sum_i h_i \cdot G_i(K_i * A_{c_0^i}^t) \mathbb{1}_{[c_1^i = j]}, \tag{5.2}$$

   where $h \in \mathbb{R}^{|K|}$ is a vector weighting the influence of kernel-growth-functions pairs (K_i, G_i) on the growth.

2. Add a fraction of U^t to the current state A^t and clip the result to get the next state A_i^{t+dt}:

$$A_i^{t+dt} = \left[A_i^t + dt \cdot U_i^t \right]_0^1.$$

Flow Lenia: Plantec, Hamon, Etcheverry, et al., 2023 propose to modify Lenia by adding a constraint on the total mass during the simulation. Flow Lenia is *mass conservative*, meaning that the sum of activations (interpreted as mass) across all cells of the grid and for each channels is

constant over time:

$$\sum_{x \in \mathcal{L}} A_c^t = \sum_{x \in \mathcal{L}} A_c^{t+dt}, \ \forall t, \forall c \in \{1, \dots, C\}.$$

In Flow Lenia, U^t is interpreted as an affinity map, and the matter contained in the grid A^t moves towards higher affinity regions by following the local gradient $\nabla U : \mathcal{L} \to \mathbb{R}^2$. To calculate the instantaneous speed of matter at timestep t, the authors define a flow $F : \mathcal{L} \to (\mathbb{R}^2)^C$, such as:

$$\begin{cases} F_i^t = (1 - \alpha^t) \nabla U_i^t - \alpha^t \nabla A_\Sigma^t \\ \alpha^t(p) = [(A_\Sigma^t(p)/\phi_A)^n]_0^1 \end{cases}$$

where $A_\Sigma^t(p) = \sum_{i=1}^C A_i^t(p)$ is the total mass in location p for all channels, and $-\nabla A_\Sigma^t$ is a diffusion term to avoid concentrating all the matter in very small regions. $\alpha : \mathcal{L} \to [0, 1]$ weights the influence of each term such that diffusion dominates in locations where the mass is close to a critical mass $\phi_A \in \mathbb{R}_{>0}$. Following authors of Flow Lenia, we mainly set $n > 1$.

Finally, to compute the next state A_i^{t+dt}, Flow Lenia moves matter in space following F using the reintegration tracking method introduced by Moroz (2020). This method can be seen as a grid-based approximation to particle systems conserving the total mass over time. To calculate the incoming matter in a cell $p \in \mathcal{L}$ at timestep t from another cell p', reintegration tracking moves the mass contained in p' to a square distribution $\mathcal{D}$ centered on $p'' = p' + dt \cdot F^t(p')$ and computes the proportion of mass arriving from p' in p as the integral:

$$I_i(p', p) = \int_{\Omega(p)} \mathcal{D}(p_i'', s)$$

where $\Omega(p)$ is the domain of cell p, and s is an hyperparameter controlling the side length of the uniform square distribution $\mathcal{D}$, acting as a form of temperature (larger s implies a broader dispersal of matter from p'). An illustrated and more detailed explanation of reintegration tracking can be found in Plantec, Hamon, Etcheverry, et al. (2023). The resulting Flow Lenia update rule for the cell p can be written as:

$$A_i^{t+dt}(p) = \sum_{p' \in \mathcal{L}} A_i^t(p') I_i(p', p).$$

In the following experiments, we use hyperparameter ranges identical to those used in the original paper.

5.3 Methods

One of the major challenges of cellular automata is to find spatially localized patterns (SLPs) that are interesting and singular compared with the average pattern distributions obtained during simulations. For instance, in Flow Lenia, most patterns (or creatures) are radially symmetrical, static and circular in shape. Interesting cases found by Plantec, Hamon, Etcheverry, et al. (2023) include worm shaped creatures, directed motion, angular motion and navigation through obstacles, among others. However, finding these interesting cases is often difficult and tediously carried out by the experimenters because there is no strict definition of what is considered as interesting, or novel. In this section, we propose a method to automatically search for interesting patterns in cellular automata, working in two steps: 1. We train an hybrid Transformer-CNN model, called Flow Lenia Transformer (FLT), learning to predict the future states of Flow Lenia from the initial state, 2. Then, we propose to use this model in a search algorithm (such as an evolutionary or QD algorithm) to drive the search process towards novel and surprising patterns.

In the following, we use an augmented version of Flow Lenia proposed by Plantec, Hamon, Etcheverry, et al. (2023) allowing to locally embed simulation parameters inside the CA. Intuitively, this allows for different parts of the same simulation to be governed by different rules. For instance, matter on a given part of the grid can obey different rules to that present in other parts, with a granularity identical to that of the activation grid A, meaning that each position p can be governed by its proper parameters, allowing creatures that might not have developed under the same set of parameters to be included and interact in the same simulation, and at the same time. Concretely, a vector of parameters is attached to every cell of the grid, forming a parameter map $P : \mathcal{L} \to \Theta$ where Θ is the parameter space. Regarding the choice of parameters to be embedded in P, we follow Plantec, Hamon, Etcheverry, et al. (2023) and chose to embed $h \in \mathcal{R}^{|K|}$, the vector weighting the importance of kernels in Equation 5.2. Thus, P locally modifies how the affinity map is computed as:

$$U_j^t(p) = \sum_{i,k} P_k^t(p) \cdot G_k(K_k * A_i^t)(p).$$

Of course, these parameter vectors are not static and move along with matter during an episode. To determine which parameter vector should be retained when several are competing for the same position p at the next timestep $t + dt$, we follow the softmax sampling method (Plantec, Hamon, Etcheverry, et al., 2023), which samples a parameter vector in the set of incoming ones with probability proportional to the respective quantities of incoming matter, as:

$$\mathbb{P}\left[P^{t+dt}(p) = P^t(x)\right] = \frac{\exp\left(A^t(x)I(x,p)\right)}{\sum_{p' \in \mathcal{L}} \exp\left(A^t(p')I(p',p)\right)}.$$

5.3.1 The Flow Lenia Transformer

The FLT is a GPT-based causal Transformer (Radford, Narasimhan, Salimans, et al., 2018) that trains on supervised[1] datasets of Flow Lenia simulated episodes and learns to predict the future of an episode simply by knowing its initial state. Following the augmented version of Flow Lenia presented in Section 5.3, an initial state is composed of the initial grid A^0, and the initial parameter map P^0. Initial patterns in A^0 are set with a 40×40 matter patch drawn from uniform distribution in the center of the grid and no matter elsewhere (as depicted in Figure 5.2), and parameters in P (i.e. kernels) are also randomly drawn from uniform distribution for every position corresponding to matter patch and null elsewhere.

To generate the dataset on which the FLT trains, we simply run multiple simulations of Flow Lenia with varying initial states (i.e. varying initial matter patches and parameter sets). In practice, to save storage space and simplify the overall training pipeline, we do not create a single massive dataset prior to training but rather generate smaller datasets on the fly during each evaluation phase[2]. Table 5.1 summarizes the Flow Lenia hyperparameter ranges used in our experiments, these are identical to those used in Plantec, Hamon, Etcheverry, et al. (2023).

Neighborhood	**Growth functions**
$R \in [2, 25]$	$\mu \in [0.05, 0.5]$ *
$r \in [0.2, 1]$ *	$\sigma \in [0.001, 0.2]$ *
Kernels	**Flow**
$h \in [0, 1]$ *	s 0.65
$a \in [0, 1]^k$ *	n 2
$b \in [0, 1]^k$ *	dt 0.2
$w \in [0.01, 0.5]^k$ *	

Table 5.1 – FLT dataset generation parameter space. Parameters marked with a * must be sampled for each kernel-growth function pair.

The architecture of the FLT is very similar to that of the Quality-Diversity Transformer (QDT) from Chapter 4, except from the fact that input embedding layers and prediction layers are implemented as CNNs, due to the nature of the inputs to be processed (activation grids and parameter maps can be thought as images). The use of CNNs both in the input and in the output layer significantly improves the model's performances over dense layer baselines.

To train the FLT, we proceed in a similar way to the QDT training and perform inference on batches of episodes drawn from the dataset. Inference is made for the whole trajectory at once, since the FLT is also causal and cannot cheat by looking at previous elements in the trajectory.

[1] Here we could almost say self-supervised training, since we are not limited with the amount of data generated, and labels are simply activations grids at timestep $t + 1$, similar to natural language processing techniques.

[2] We keep all simulation hyperparameters equal (except for embedded parameters in P) to ensure that we train on the same data distribution overall.

A trajectory τ is constructed as follows:

$$\tau = (A^0, P^0, A^1, P^1, ..., A^T, P^T).$$

At any given timestep t, the FLT takes as input the activation grid A^t and the parameter map P^t and predicts $\hat{A}^{t+1}$ and $\hat{P}^{t+1}$. To train the model, we use a mean squared error (MSE) loss for both activation grids and parameter maps. The FLT loss over a single episode is given by:

$$L = \frac{1}{T}\sum_{t=1}^{T}\left(\left\|A_t - \hat{A}_t\right\|_2^2 + \left\|P_t - \hat{P}_t\right\|_2^2\right). \tag{5.3}$$

To evaluate the Flow Lenia Transformer, we simply generate an initial state (A^0, P^0), and autoregressively unroll a whole episode without further clue for the model. We then compare the frames produced by the FLT to those of the real episode which started from the same initial state (A^0, P^0). This exercise is difficult because, having no checkpoints throughout the episode, the model can quickly accumulate errors and deviate widely from ground truth. Figure 5.2 illustrates the evaluation process of the FLT.

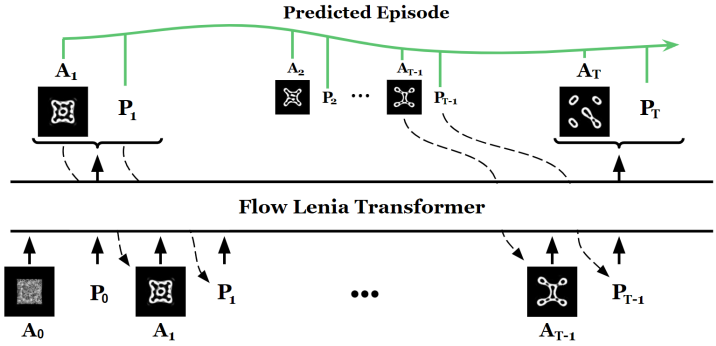


Figure 5.2 – Evaluation process of the FLT. The model autoregressively produces an entire episode based on an initial state made of initial activations A^0 and initial parameters map P^0.

5.3.2 Searching for Interesting Patterns

At the time of writing, this part is still a work in progress. Inspired by surprise-driven algorithms for divergent search (Gravina, Liapis, and Yannakakis, 2016; Yannakakis and Liapis, 2016; Gravina, Liapis, and Yannakakis, 2018), we propose to use a trained FLT to quantify the magnitude of surprise caused by new creatures generated in search algorithms (such as evolutionary algorithms). The central idea of surprise-driven algorithms is to encourage exploration by focusing on elements that produce results that are unexpected compared to current expectations (Burda, Edwards, Storkey, et al., 2018). These expectations can be based on predictive models such as the FLT. When an observation contradicts what was expected, it is marked as surprising,

and the search algorithm is encouraged to explore that region of the solution space further. Perhaps more simply, a surprise-based model could also serve as a passive tool that evaluates patterns generated by a search algorithm, and stores them for further examination by the experimenter when the surprise exceeds a certain threshold.

In our case, using the error of the FLT trying to predict the evolution of a Flow Lenia simulation, either by comparing $\hat{A}^T$ to ground truth A^T or by considering any other time window, could help automate the task of finding interesting creatures.

5.4 Preliminary Results

In this section, we present some early results obtained by evaluating the FLT. As mentioned in Section 5.3.1, the model is evaluated by autoregressively generating whole episodes, starting from an initial state (A^0, P^0). This task is particularly difficult for the model because it has to predict the evolution of the simulation with only the information contained in the initial state, and an early error in prediction can lead to an accumulation of errors that is impossible to correct. Figure 5.3 presents some cherry picked evaluations of the FLT. For each of the three

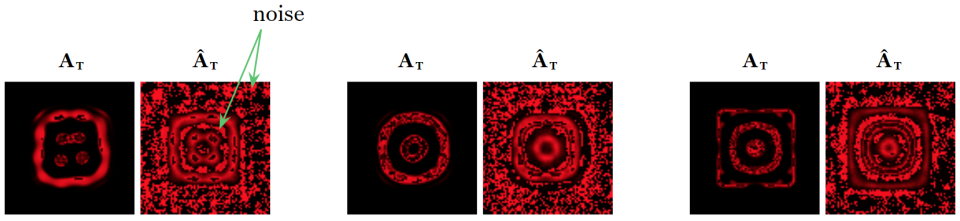


Figure 5.3 – Evaluation results of the FLT over three different examples, each starting from different (random) initial states. For each example, the frames represent the ground truth last activation map (A^T) and the predicted last activation map $(\hat{A}^T)$.

examples, the ground truth label of the activation grid at the end of the episode is on the left, while the model prediction is on the right. Although the predictions are highly noisy, we can see that the model is still able to predict the overall pattern structure emerging at the end of the episode. In (a), the predicted central pattern resembles the label and also contains the four small dots —amidst the noise—, in (b) the prediction does have the hollow ring contained within a larger circle, and in (c) the predicted outer structure of the pattern is square in shape, as on the ground truth label. These early results, while being far from truly satisfying, are encouraging regarding the model's ability to predict the final result of a simulation from a single initial state.

Figure 5.4 shows chronological results of example (a) from Figure 5.3. As an episode lasts for 50 timesteps, we provide frames for $t = 1$, 15, 30, 45 and 50. To better show the pattern

predicted by the FLT, we manually remove the noise, depicting a result we aim to achieve in the continuation of this work. To a certain degree of accuracy, the FLT is able to predict and

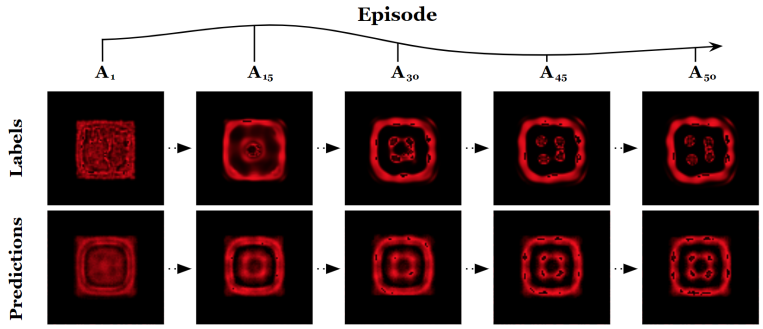
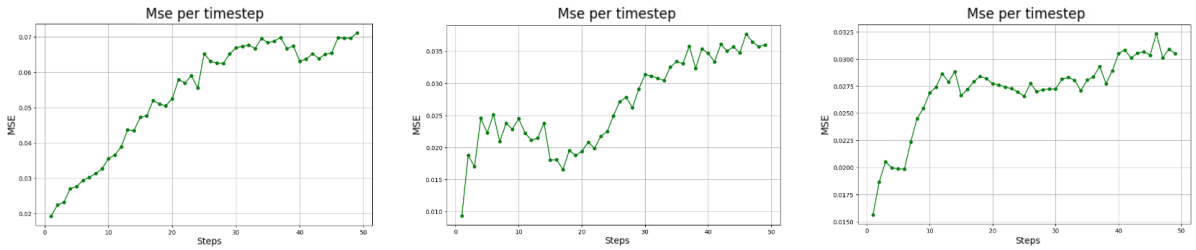


Figure 5.4 – Chronology of labels and predictions over the entire episode from Figure 5.3a. Prediction frames are denoised manually to better show the central pattern.

reproduce the evolution of the true pattern over time. Note that its capacity to accurately predict each frame of an episode stems from the nature of the loss given in Equation 5.3, however, this is not a necessary property for the FLT to be used in a search algorithm, as one may be interested only in the final state.

Finally, we confirm that our model tends to accumulate error over the course of an evaluation episode, due to autoregressive prediction. Figure 5.5 gives the MSE *per timestep* (as given in Equation 5.3 but without the mean calculation) for all three examples depicted in Figure 5.3. In all cases, error tends to increase as the episode progresses.



(a) (b) (c)

Figure 5.5 – Mean square error per timestep for all three examples (a), (b) and (c) presented in Figure 5.3, respectively.

Chapter 6

Conclusion

Life is this crazy, mystical thing,
and sometimes you just go out like a buster.
Mang0.

There is no point in claiming that this thesis followed a meticulously predetermined plan, with methods carefully devised to answer a neatly defined initial problem. Research, as many who have come down this path would agree, often takes unexpected turns, and what sometimes seems as a clear goal may evolve into something different. Rather, this thesis studied three main topics revolving around divergent search (or exploration) through diversity, and proposed original methods for addressing them. In this final section, we intend to provide details not mentioned in scientific publications, personal reflections and critical conclusions about each contribution chapter.

Sample efficient diversity search with reinforcement learning. Chapter 3 presents QD-PG, a hybrid quality-diversity/policy-gradient algorithm combining the exploration capabilities of QD methods and the sample efficiency of reinforcement learning methods. QD-PG features a double exploration mechanism thanks to 1. the structural exploration mechanism from the MAP-Elites grid and its associated random selection operator, and 2. the diversity policy gradient, which exploits the estimated gradient of the objective function and the analytical structure of neural network policies to open the black box and push towards novelty rapidly. In practice, we like to think of QD-PG as a formula 1, i.e., an algorithm that does its job extremely well when the right conditions are met, but that performs poorly when these conditions slightly change —there is a speed bump in the road. QD-PG inherits *some* of the shortcomings of policy gradient methods, making it very sensitive to hyperparameter changes, and thus less robust. In fact, most of the effort invested during its development went into finding a set of hyperparameters that made it stable in Ant-Omni, whose difficulty as an environment is often underestimated because the maze —and the 3000 steps time horizon— is disproportionately

large when compared to other standard continuous control environments. Finally, the overall algorithmic structure of QD-PG is arbitrary and has originally been tested with other archive mechanisms such as a Pareto front (with relatively poorer results), but the idea of a diversity policy gradient is interesting, original and worth taking up in future work.

Distillation of a repertoire of diverse policies into a unique Transformer. Chapter 4 can be thought, in some sense, as the next step of the quality-diversity optimization process. We have produced a repertoire of diverse solutions for a given problem, but now what do we do with it? In this chapter, we first take a detour to propose a variant of MAP-Elites that produces solutions not subject to the variance and undesirable behavior of classical MAP-Elites solutions. ME-LS is a simple yet robust and powerful algorithm producing steady and regular solutions in the behavior space. We argue that additional evaluations in ME-LS hardly make it less sample efficient because 1. the algorithm converges roughly as fast as MAP-Elites, and 2. it converges to values that are closer to the true capabilities of the final repertoire when re-evaluated, whereas MAP-Elites is unable to reproduce the training performances. In practice, ME-LS and the data filtering step really are the key components that made the QDT work properly. Indeed, it is the data quality used for the supervised training phase of the QDT that made all the difference between an intermediate, relatively imprecise model and an accurate one that is able to cover the whole behavior space and generalize to some extent. Throughout this work, we found that the architecture details of the QDT have little importance as long as it makes sense. The use of a transformer is an obvious choice, given its ability to handle sequential data and the recent release of the Decision Transformer, but perhaps any other sequential model would have done the trick. Now starting from a trained QDT —a behavior-conditioned policy that is compact (more compact than the sum of all policies in the MAP-Elites grid) and versatile— there is plenty of room to build on top of it in future work, or to incorporate it into more complex systems.

Learning continuous cellular automata world model with Transformers. Chapter 5 presents an early work around the automatic search for interesting patterns in Flow Lenia. Only the first part, which involves learning a model of the Flow Lenia world using a Transformer model, has so far been studied. Preliminary results are encouraging and demonstrate that predicting the final state of a continuous cellular automaton, only with the initial state and without knowledge of the rules governing it, seems possible to a certain extent. So far, the most challenging part about this chapter has been the technical aspects and constraints linked to compute resources memory and the model size. We hypothesize —and seem to observe— that larger models help to predict Flow Lenia episodes more accurately, but infrastructure constraints has so far obliged us to run this work on 8GB GPUs/TPUs, which can only accommodate relatively small models. There are several ways around these constraints, such as mixed precision training and model sharding.

Bibliography

Abramson, Josh, Jonas Adler, Jack Dunger, Richard Evans, Tim Green, Alexander Pritzel, Olaf Ronneberger, Lindsay Willmore, Andrew J Ballard, Joshua Bambrick, et al. (2024). Accurate structure prediction of biomolecular interactions with AlphaFold 3. *Nature*, pp. 1–3.

Achiam, Josh, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Alvarez, Alberto, Steve Dahlskog, Jose Font, and Julian Togelius (2019). Empowering quality diversity in dungeon design with interactive constrained MAP-Elites. **In** *2019 IEEE Conference on Games (CoG)*. IEEE, pp. 1–8.

Amin, Susan, Maziar Gomrokchi, Harsh Satija, Herke Van Hoof, and Doina Precup (2021). A survey of exploration methods in reinforcement learning. *arXiv preprint arXiv:2109.00157*.

Arulkumaran, Kai, Antoine Cully, and Julian Togelius (2019). Alphastar: An evolutionary computation perspective. **In** *Proceedings of the genetic and evolutionary computation conference companion*, pp. 314–315.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E Hinton (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bachute, Mrinal R and Javed M Subhedar (2021). Autonomous driving architectures: insights of machine learning and deep learning algorithms. *Machine Learning with Applications* 6, p. 100164.

Barto, Andrew G, Richard S Sutton, and Charles W Anderson (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics* 5, pp. 834–846.

Bellemare, Marc G, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C Machado, Subhodeep Moitra, Sameera S Ponda, and Ziyu Wang (2020). Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature* 588.7836, pp. 77–82.

Bellman, Richard (1952). On the theory of dynamic programming. *Proceedings of the national Academy of Sciences* 38.8, pp. 716–719.

Bellman, Richard (1966). Dynamic programming. *science* 153.3731, pp. 34–37.

Berlekamp, Elwyn R, John H Conway, and Richard K Guy (2004). *Winning ways for your mathematical plays, volume 4*. AK Peters/CRC Press.

Bertschinger, Edmund (1998). Simulations of structure formation in the universe. *Annual Review of Astronomy and Astrophysics* 36.1, pp. 599–654.

Bojarski, Mariusz, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Bottou, Léon (2010). Large-scale machine learning with stochastic gradient descent. **In** *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*. Springer, pp. 177–186.

Bradbury, James, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang (2018). *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13.

Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems* 33, pp. 1877–1901.

Burda, Yuri, Harrison Edwards, Amos Storkey, and Oleg Klimov (2018). Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*.

Carion, Nicolas, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko (2020). End-to-end object detection with transformers. **In** *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*. Springer, pp. 213–229.

Cazenille, Leo, Nicolas Bredeche, and Nathanael Aubert-Kato (2019). Exploring Self-Assembling Behaviors in a Swarm of Bio-micro-robots using Surrogate-Assisted MAP-Elites. *arXiv preprint arXiv:1910.00230*.

Chalumeau, Felix, Raphael Boige, Bryan Lim, Valentin Macé, Maxime Allard, Arthur Flajolet, Antoine Cully, and Thomas Pierrot (2022). Neuroevolution is a competitive alternative to reinforcement learning for skill discovery. *arXiv preprint arXiv:2210.03516*.

Chalumeau, Felix, Bryan Lim, Raphael Boige, Maxime Allard, Luca Grillotti, Manon Flageat, Valentin Macé, Guillaume Richard, Arthur Flajolet, Thomas Pierrot, et al. (2024). QDax: A library for quality-diversity and population-based algorithms with hardware acceleration. *Journal of Machine Learning Research* 25.108, pp. 1–16.

Chalumeau, Felix, Thomas Pierrot, Valentin Macé, Arthur Flajolet, Karim Beguir, Antoine Cully, and Nicolas Perrin-Gilbert (2022). Assessing Quality-Diversity Neuro-Evolution Algorithms Performance in Hard Exploration Problems. *arXiv preprint arXiv:2211.13742*.

Chan, Bert Wang-Chak (2018). Lenia-biology of artificial life. *arXiv preprint arXiv:1812.05433*.

Chatzilygeroudis, Konstantinos, Antoine Cully, Vassilis Vassiliades, and Jean-Baptiste Mouret (2021). Quality-diversity optimization: a novel branch of stochastic optimization. **In** *Black Box Optimization, Machine Learning, and No-Free Lunch Theorems*. Springer, pp. 109–135.

Chen, Lili, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch (2021). Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems* 34, pp. 15084–15097.

Chowdhery, Aakanksha, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. (2023). Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research* 24.240, pp. 1–113.

Clune, Jeff, Jean-Baptiste Mouret, and Hod Lipson (2013). The evolutionary origins of modularity. *Proceedings of the Royal Society b: Biological sciences* 280.1755, p. 20122863.

Colas, Cédric, Vashisht Madhavan, Joost Huizinga, and Jeff Clune (2020). Scaling map-elites to deep neuroevolution. **In** *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 67–75.

Colas, Cédric, Olivier Sigaud, and Pierre-Yves Oudeyer (2018). GEP-PG: Decoupling exploration and exploitation in deep reinforcement learning algorithms. **In** *International conference on machine learning*. PMLR, pp. 1039–1048.

Conti, Edoardo, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth Stanley, and Jeff Clune (2018). Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. **In** *Advances in neural information processing systems*, pp. 5027–5038.

Cully, Antoine, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret (2015). Robots that can adapt like animals. *Nature* 521.7553, pp. 503–507.

Cully, Antoine and Yiannis Demiris (2017). Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation* 22.2, pp. 245–259.

Cully, Antoine and Yiannis Demiris (2018). Hierarchical behavioral repertoires with unsupervised descriptors. **In** *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 69–76.

Degrave, Jonas, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. (2022). Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature* 602.7897, pp. 414–419.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Doan, Thang, Bogdan Mazoure, Audrey Durand, Joelle Pineau, and R Devon Hjelm (2019). Attraction-Repulsion Actor-Critic for Continuous Control Reinforcement Learning. *arXiv preprint arXiv:1909.07543*.

Doncieux, Stephane, Alban Laflaquière, and Alexandre Coninx (2019). Novelty search: a theoretical perspective. **In** *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 99–106.

Dong, Linhao, Shuang Xu, and Bo Xu (2018). Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. **In** *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, pp. 5884–5888.

Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Drossel, Barbara and Franz Schwabl (1992). Self-organized critical forest-fire model. *Physical review letters* 69.11, p. 1629.

Duchi, John, Elad Hazan, and Yoram Singer (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* 12.7.

Ecoffet, Adrien, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune (2019). Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*.

Engebraaten, Sondre A, Jonas Moen, Oleg A Yakimenko, and Kyrre Glette (2020). A framework for automatic behavior generation in multi-function swarms. *Frontiers in Robotics and AI* 7, p. 579403.

Ermentrout, G Bard and Leah Edelstein-Keshet (1993). Cellular automata approaches to biological modeling. *Journal of theoretical Biology* 160.1, pp. 97–133.

Espinasse, Bernard, Sébastien Fournier, Adrian Chifu, Gaël Guibon, René Azcurra, and Valentin Mace (2019). On the Use of Dependencies in Relation Classification of Text with Deep Learning. **In** *International Conference on Computational Linguistics and Intelligent Text Processing*. Springer, pp. 379–391.

Evans, Simon, Jim Gao, Dave Anderson, and Sam Rawal (2018). DeepMind AI Reduces Google Data Centre Cooling Bill by 40%. *DeepMind Blog*.

Eysenbach, Benjamin, Abhishek Gupta, Julian Ibarz, and Sergey Levine (2018). Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*.

Feynman, Richard P (2018). Simulating physics with computers. **In** *Feynman and computation*. CRC Press, pp. 133–153.

Flageat, Manon, Felix Chalumeau, and Antoine Cully (2022). Empirical analysis of PGA-MAP-Elites for Neuroevolution in Uncertain Domains. *ACM Transactions on Evolutionary Learning*.

Flageat, Manon and Antoine Cully (2020). Fast and stable MAP-Elites in noisy domains using deep grids. **In** *Artificial Life Conference Proceedings 32*. MIT Press, pp. 273–282.

Flageat, Manon and Antoine Cully (2023). Uncertain Quality-Diversity: Evaluation methodology and new methods for Quality-Diversity in Uncertain Domains. *arXiv preprint arXiv:2302.00463*.

Flageat, Manon, Bryan Lim, Luca Grillotti, Maxime Allard, Simón C Smith, and Antoine Cully (2022). Benchmarking Quality-Diversity Algorithms on Neuroevolution for Reinforcement Learning. *arXiv preprint arXiv:2211.02193*.

Flet-Berliac, Yannis, Johan Ferret, Olivier Pietquin, Philippe Preux, and Matthieu Geist (2021). Adversarially guided actor-critic. *arXiv preprint arXiv:2102.04376*.

Fogel, David B (1998). *Artificial intelligence through simulated evolution*. Wiley-IEEE Press.

Fontaine, Matthew and Stefanos Nikolaidis (2021). Differentiable quality diversity. *Advances in Neural Information Processing Systems* 34, pp. 10040–10052.

Fontaine, Matthew C, Julian Togelius, Stefanos Nikolaidis, and Amy K Hoover (2020). Covariance matrix adaptation for the rapid illumination of behavior space. **In** *Proceedings of the 2020 genetic and evolutionary computation conference*, pp. 94–102.

Forestier, Sébastien, Yoan Mollard, and Pierre-Yves Oudeyer (2017). Intrinsically motivated goal exploration processes with automatic curriculum learning. *arXiv preprint arXiv:1708.02190*.

Frans, Kevin, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman (2018). Meta learning shared hierarchies. *Proc. of ICLR*.

Freeman, C Daniel, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem (2021). Brax–A Differentiable Physics Engine for Large Scale Rigid Body Simulation. *arXiv preprint arXiv:2106.13281*.

Frisch, Uriel, Brosl Hasslacher, and Yves Pomeau (1986). Lattice-gas automata for the Navier-Stokes equation. *Physical review letters* 56.14, p. 1505.

Fujimoto, Scott, Herke Van Hoof, and David Meger (2018). Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*.

Gardner, Martin (1970). Mathematical Games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American* 223.4, pp. 120–123.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). Deep feedforward networks. *Deep learning* 1.

Gravina, Daniele, Ahmed Khalifa, Antonios Liapis, Julian Togelius, and Georgios N Yannakakis (2019). Procedural content generation through quality diversity. **In** *2019 IEEE Conference on Games (CoG)*. IEEE, pp. 1–8.

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

Gravina, Daniele, Antonios Liapis, and Georgios N Yannakakis (2018). Quality diversity through surprise. *IEEE Transactions on Evolutionary Computation* 23.4, pp. 603–616.

Haarnoja, Tuomas, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. (2018). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

Harris, Charles R., K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant (2020). Array programming with NumPy. *Nature* 585.7825, pp. 357–362.

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

Hornby, Gregory, Al Globus, Derek Linden, and Jason Lohn (2006). Automated antenna design with evolutionary algorithms. **In** *Space 2006*, p. 7242.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). Multilayer feedforward networks are universal approximators. *Neural networks* 2.5, pp. 359–366.

Howard, Ronald A (1960). Dynamic programming and markov processes.

Islam, Riashat, Zafarali Ahmed, and Doina Precup (2019). Marginalized State Distribution Entropy Regularization in Policy Optimization. *arXiv preprint arXiv:1912.05128*.

Jaderberg, Max, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. (2017). Population-based training of neural networks. *arXiv preprint arXiv:1711.09846*.

Janner, Michael, Qiyang Li, and Sergey Levine (2021). Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems* 34, pp. 1273–1286.

Jegorova, Marija, Stéphane Doncieux, and Timothy M Hospedales (2020). Behavioral repertoire via generative adversarial policy networks. *IEEE Transactions on Cognitive and Developmental Systems*.

Jumper, John, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature* 596.7873, pp. 583–589.

Jung, Whiyoung, Giseung Park, and Youngchul Sung (2020). Population-Guided Parallel Policy Search for Reinforcement Learning. **In** *International Conference on Learning Representations*.

Jurgovsky, Johannes, Michael Granitzer, Konstantin Ziegler, Sylvie Calabretto, Pierre-Edouard Portier, Liyun He-Guelton, and Olivier Caelen (2018). Sequence classification for credit-card fraud detection. *Expert systems with applications* 100, pp. 234–245.

Justesen, Niels, Sebastian Risi, and Jean-Baptiste Mouret (2019). Map-elites for noisy domains by adaptive sampling. **In** *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 121–122.

Khadka, Shauharda, Somdeb Majumdar, Santiago Miret, Evren Tumer, Tarek Nassar, Zach Dwiel, Yinyin Liu, and Kagan Tumer (2019). Collaborative evolutionary reinforcement learning. *arXiv preprint arXiv:1905.00976*.

Khadka, Shauharda and Kagan Tumer (2018). Evolution-Guided Policy Gradient in Reinforcement Learning. **In** *Neural Information Processing Systems*.

Kingma, Diederik P and Jimmy Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Koos, Sylvain, Jean-Baptiste Mouret, and Stéphane Doncieux (2012). The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation* 17.1, pp. 122–145.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25.

Kumar, Saurabh, Aviral Kumar, Sergey Levine, and Chelsea Finn (2020). One solution is not all you need: Few-shot extrapolation via structured maxent rl. *Advances in Neural Information Processing Systems* 33, pp. 8198–8210.

LeCun, Yann (2018). *Deep Learning is Dead. Vive Differentiable Programming!* Facebook post, January 5, 2018. https://www.facebook.com/yann.lecun/posts/10155003011462143.

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Lee, Kuang-Huei, Ofir Nachum, Sherry Yang, Lisa Lee, C. Daniel Freeman, Sergio Guadarrama, Ian Fischer, Winnie Xu, Eric Jang, Henryk Michalewski, and Igor Mordatch (2022). Multi-

Game Decision Transformers. **In** *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho.

Lee, Lisa, Benjamin Eysenbach, Emilio Parisotto, Eric Xing, Sergey Levine, and Ruslan Salakhutdinov (2019). Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*.

Lehman, Joel, Jeff Clune, Dusan Misevic, Christoph Adami, Lee Altenberg, Julie Beaulieu, Peter J Bentley, Samuel Bernard, Guillaume Beslon, David M Bryson, et al. (2020). The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities. *Artificial life* 26.2, pp. 274–306.

Lehman, Joel and Kenneth O Stanley (2011a). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation* 19.2, pp. 189–223.

Lehman, Joel and Kenneth O Stanley (2011b). 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.

Levine, Sergey, Chelsea Finn, Trevor Darrell, and Pieter Abbeel (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research* 17.39, pp. 1–40.

Lian, Yongsheng, Akira Oyama, and Meng-Sing Liou (2010). Progress in design optimization using evolutionary algorithms for aerodynamic problems. *Progress in Aerospace Sciences* 46.5-6, pp. 199–223.

Lillicrap, Timothy P, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Macé, Valentin, Raphaël Boige, Felix Chalumeau, Thomas Pierrot, Guillaume Richard, and Nicolas Perrin-Gilbert (2023). The quality-diversity transformer: Generating behavior-conditioned trajectories with decision transformers. **In** *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1221–1229.

Macé, Valentin and Christophe Servan (2019). Using Whole Document Context in Neural Machine Translation. **In** *Proceedings of the 16th International Conference on Spoken Language Translation*. Hong Kong: Association for Computational Linguistics.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org.

McCulloch, Warren S and Walter Pitts (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* 5, pp. 115–133.

McKinney, Scott Mayer, Marcin Sieniek, Varun Godbole, Jonathan Godwin, Natasha Antropova, Hutan Ashrafian, Trevor Back, Mary Chesus, Greg S Corrado, Ara Darzi, et al. (2020).

International evaluation of an AI system for breast cancer screening. *Nature* 577.7788, pp. 89–94.

Message Passing Interface Forum (2021). *MPI: A Message-Passing Interface Standard Version 4.0*.

Metropolis, Nicholas and Stanislaw Ulam (1949). The monte carlo method. *Journal of the American statistical association* 44.247, pp. 335–341.

Mordvintsev, Alexander, Eyvind Niklasson, and Ettore Randazzo (2023). *Particle Lenia*. https://google-research.github.io/self-organising-systems/particle-lenia/.

Moroz, Michael (2020). *Reintegration Tracking*. https://michaelmoroz.github.io/Reintegration-Tracking/.

Morrison, Douglas, Peter Corke, and Jurgen Leitner (2020). EGAD! an Evolved Grasping Analysis Dataset for diversity and reproducibility in robotic manipulation. *IEEE Robotics and Automation Letters*.

Mouret, Jean-Baptiste (2020). Evolving the behavior of machines: from micro to macroevolution. *Iscience* 23.11.

Mouret, Jean-Baptiste and Jeff Clune (2015). Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*.

Nair, Vinod and Geoffrey E Hinton (2010). Rectified linear units improve restricted boltzmann machines. **In** *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.

Nasiriany, Soroush, Vitchyr H Pong, Steven Lin, and Sergey Levine (2019). Planning with goal-conditioned policies. *arXiv preprint arXiv:1911.08453*.

Nilsson, Olle and Antoine Cully (2021). Policy gradient assisted map-elites. **In** *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 866–875.

Parker-Holder, Jack, Aldo Pacchiano, Krzysztof Choromanski, and Stephen Roberts (2020). Effective Diversity in Population-Based Reinforcement Learning. **In** *Neural Information Processing Systems*.

Paszke, Adam, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer (2017). Automatic differentiation in PyTorch.

Pierrot, Thomas, Valentin Macé, Felix Chalumeau, Arthur Flajolet, Geoffrey Cideron, Karim Beguir, Antoine Cully, Olivier Sigaud, and Nicolas Perrin-Gilbert (2022). Diversity policy gradient for sample efficient quality-diversity optimization. **In** *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1075–1083.

Pierrot, Thomas, Valentin Macé, Jean-Baptiste Sevestre, Louis Monier, Alexandre Laterre, Nicolas Perrin, Karim Beguir, and Olivier Sigaud (2021). Factored action spaces in deep reinforcement learning.

Pierrot, Thomas, Guillaume Richard, Karim Beguir, and Antoine Cully (2022). Multi-Objective Quality Diversity Optimization. *arXiv preprint arXiv:2202.03057*.

Plantec, Erwan, Gautier Hamon, Mayalen Etcheverry, Pierre-Yves Oudeyer, Clément Moulin-Frier, and Bert Wang-Chak Chan (2023). Flow-Lenia: Towards open-ended evolution in

cellular automata through mass conservation and parameter localization. **In** *Artificial Life Conference Proceedings 35*. Vol. 2023. 1. MIT Press, p. 131.

Polyak, Boris T (1964). Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics* 4.5, pp. 1–17.

Pong, Vitchyr H, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine (2019). Skew-fit: State-covering self-supervised reinforcement learning. *arXiv preprint arXiv:1903.03698*.

Pourchot, Aloıs and Olivier Sigaud (2018). CEM-RL: Combining evolutionary and gradient-based methods for policy search. *arXiv preprint arXiv:1810.01222*.

Pugh, Justin K, Lisa B Soros, and Kenneth O Stanley (2016). Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI* 3, p. 202845.

Radford, Alec, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. (2018). Improving language understanding by generative pre-training.

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog* 1.8, p. 9.

Rafler, Stephan (2011). Generalization of Conway's" Game of Life" to a continuous domain-SmoothLife. *arXiv preprint arXiv:1111.1567*.

Rakicevic, Nemanja, Antoine Cully, and Petar Kormushev (2021). Policy manifold search: Exploring the manifold hypothesis for diversity-based neuroevolution. **In** *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 901–909.

Reed, Scott, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. (2022). A generalist agent. *arXiv preprint arXiv:2205.06175*.

Rendell, Paul (2002). Turing universality of the game of life. **In** *Collision-based computing*. Springer, pp. 513–539.

Robbins, Herbert and Sutton Monro (1951). A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). Learning representations by back-propagating errors. *Nature* 323.6088, pp. 533–536.

Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision* 115, pp. 211–252.

Salimans, Tim, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sharma, Archit, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman (2019). Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*.

Shi, Longxiang, Shijian Li, Qian Zheng, Min Yao, and Gang Pan (2020). Efficient Novelty Search Through Deep Reinforcement Learning. *IEEE Access* 8, pp. 128809–128818.

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature* 529.7587, pp. 484–489.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller (2014). Deterministic policy gradient algorithms. **In** *Proceedings of the 30th International Conference in Machine Learning*.

Skinner, BF (1938). *The behavior of organisms: an experimental analysis.* Appleton-Century.

Stanton, Christopher and Jeff Clune (2016). Curiosity search: producing generalists by encouraging individuals to continually explore and acquire skills throughout their lifetime. *PloS one* 11.9, e0162235.

Sudhakaran, Shyam, Miguel González-Duque, Matthias Freiberger, Claire Glanois, Elias Najarro, and Sebastian Risi (2024). Mariogpt: Open-ended text2level generation through large language models. *Advances in Neural Information Processing Systems* 36.

Sutton, Richard S (1988). Learning to predict by the methods of temporal differences. *Machine learning* 3, pp. 9–44.

Sutton, Richard S and Andrew G Barto (1998). *Reinforcement Learning: An Introduction*.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, Richard S, David A McAllester, Satinder P Singh, Yishay Mansour, et al. (1999). Policy gradient methods for reinforcement learning with function approximation. **In** *NIPs*. Vol. 99. Citeseer, pp. 1057–1063.

Thorndike, Edward L (1898). Animal intelligence: An experimental study of the associative processes in animals. *The Psychological Review: Monograph Supplements* 2.4, p. i.

Thorndike, Edward L (1927). The law of effect. *The American journal of psychology* 39.1/4, pp. 212–222.

Tieleman, Tijmen and Geoffrey Hinton (2012). Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. *COURSERA Neural Networks Mach. Learn* 17.

Vassiliades, Vassilis, Konstantinos I. Chatzilygeroudis, and Jean-Baptiste Mouret (2016). Scaling Up MAP-Elites Using Centroidal Voronoi Tessellations. *CoRR* abs/1610.05729.

Vassiliades, Vassilis and Jean-Baptiste Mouret (Apr. 2018). Discovering the Elite Hypervolume by Leveraging Interspecies Correlation. *GECCO 2018 - Proceedings of the 2018 Genetic and Evolutionary Computation Conference*, pp. 149–156.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). Attention is all you need. *Advances in neural information processing systems* 30.

Vemula, Anirudh, Wen Sun, and J Bagnell (2019). Contrasting exploration in parameter and action space: A zeroth-order optimization perspective. **In** *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 2926–2935.

Vinyals, Oriol, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575.7782, pp. 350–354.

Watkins, Christopher JCH and Peter Dayan (1992). Q-learning. *Machine learning* 8, pp. 279–292.

Williams, Ronald J (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, pp. 229–256.

Wolfram, Stephen and M Gad-el-Hak (2003). A new kind of science. *Appl. Mech. Rev.* 56.2, B18–B19.

Yannakakis, Georgios N and Antonios Liapis (2016). Searching for surprise. **In** ICCC.

Yin, Zhiqi, Zeshi Yang, Michiel Van De Panne, and KangKang Yin (2021). Discovering diverse athletic jumping strategies. *ACM Transactions on Graphics (TOG)* 40.4, pp. 1–17.

List of Figures

List of Algorithms

List of Tables