Imperial College London

IMPERIAL COLLEGE LONDON

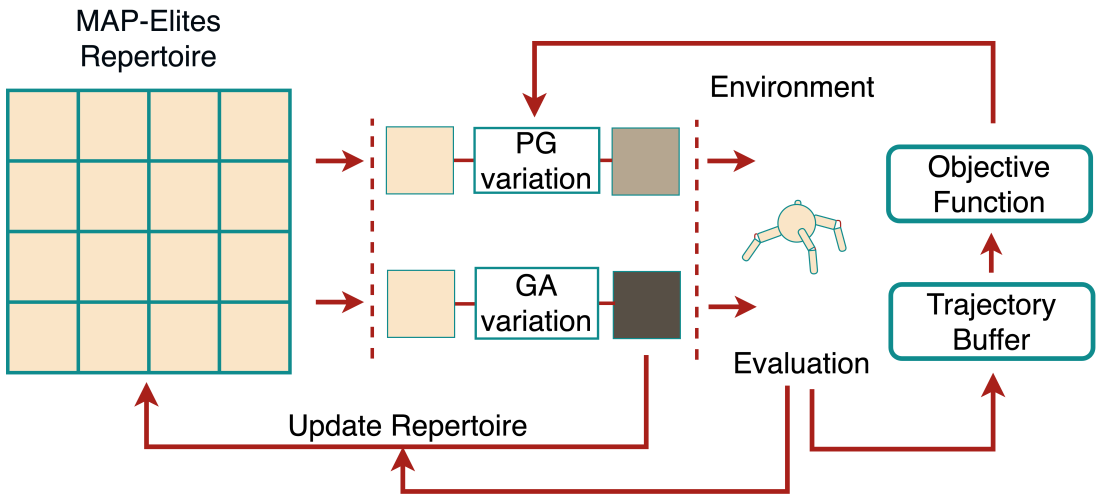
DEPARTMENT OF COMPUTING

Combining Deep Reinforcement Learning and Quality-Diversity Algorithms

Author:
Konstantinos Mitsides

Supervisor:
Dr Antoine Cully

September 2024



Abstract

This work explores the integration of Monte Carlo Policy Gradient (MCPG) methods with MAP-Elites to enhance Quality-Diversity (QD) optimisation, a family of evolutionary algorithms designed to produce a diverse set of high-performing solutions. Traditional QD optimisation, such as MAP-Elites, is driven by Genetic Algorithms and often struggles to evolve neural networks with numerous parameters and is inefficient in navigating the search space of the optimisation problem. Additionally, actor-critic based QD algorithms, like PGA-MAP-Elites and DCG-MAP-Elites-AI, while capable of handling more complex models efficiently, suffer from slow execution times and are heavily dependent on the effectiveness of the actor-critic training, which compromises scalability. Addressing these challenges, we introduce the Monte Carlo Policy Gradient MAP-Elites (MCPG-ME) algorithm, which utilises MCPG in an off-policy manner. This novel approach allows MCPG-ME to independently optimise the solutions without relying on any actor-critic training, enhancing scalability, runtime efficiency while maintaining competitive sample efficiency. Our evaluations across various continuous control locomotion tasks demonstrate that MCPG-ME excels in finding a diverse set of solutions, achieving equal or higher diversity score (coverage) than all competitors in all tasks. Additionally, it surpass the performance of the state-of-the-art algorithm, DCG-MAP-Elites-AI, in some of the tasks and consistently outperform all the other baselines in most of the tasks. Furthermore, MCPG-ME achieves high execution speeds, operating significantly faster than the actor-critic based QD algorithms, in some cases running up to nine times faster. Lastly, it demonstrates promising scalability capabilities when subjected to massive parallelisation.

Acknowledgments

I would like to thank my primary supervisor Dr. Antoine Cully for his valuable guidance and support throughout this project, his numerous pieces of advice on how to be a good researcher, and for creating a welcoming and inspirational environment in his lab. I would also like to express my gratitude to my secondary supervisor, Maxence Faldor, for making this project as smooth as possible and for helping me to work more efficiently. In general, I am grateful to everyone in the Adaptive & Intelligent Robotics Lab for being so friendly and always willing to help me and the other Master's students.

I would also like to thank my friends for their support, encouragement, and love throughout this journey. Most importantly, I am deeply grateful to my family for their unconditional love and support. Special thanks go to my sister, who has been a constant source of strength for me.

Contents

Chapter 1

Introduction

Diversity is the essence of resilience and innovation. In nature, the survival and thriving of species often rely not on their dominance in a single trait but on their ability to adapt in various ways to their surroundings [1] [2]. Inspired by this, Quality-Diversity (QD) optimisation is a family of evolutionary algorithms that aim to generate a set of both diverse and high-performing solutions to a problem [3] [4] [5]. QD optimisation localises competition among solutions by focusing on those that are similar based on some predefined or not predefined [6] characteristics. Therefore, unlike traditional optimisation methods that yield a single optimal solution, the goal of QD algorithms is to illuminate a search space of interest called the *behavioural descriptor space*, by finding the highest-performing solution per *descriptor*—the predefined characteristic(s) used to categorise and compare solutions.

In general, when optimising a model for a non-convex objective function, an algorithm that greedily follows the gradient of the objective might get stuck in local optima, thereby missing potentially better solutions. One can think of an agent navigating a maze filled with false exits. An algorithm that directs the agent to always move towards the closest apparent exit might lead the agent to a dead-end, missing the true exit that requires a more circuitous path through the maze. In contrast, QD optimisation avoids these pitfalls by simultaneously searching for multiple high-performing, diverse solutions. This approach can help identify 'stepping stones' that overcome local optima and lead to globally more optimal solutions [7] [8]. Furthermore, QD's capability to explore a variety of solutions makes it highly adaptable in dynamic environments, such as robotics, where it can develop a range of behaviours. This adaptability proves invaluable when a robot's environment changes or it sustains damage, enabling it to switch to alternative strategies [9] [10] [11]. Lastly, the diversity inherent in QD optimisation has been shown to facilitate more effective exploration in scenarios where reward signals are sparse or deceptive [12] [13], enhancing the direct optimisation of objectives.

The Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) [14] is one of the most widely known QD optimisation algorithms, designed to explore and illuminate the diversity of high-performing solutions across a multidimensional search space. It traditionally relies on Genetic Algorithms (GAs), which are favoured for their ability to diversify the search through mechanisms such as mutation and crossover. However, this reliance on GAs comes with some limitations. Specifically, it confines MAP-Elites to low-dimensional problems, making it inadequate for evolving neural networks with a large number of parameters [9] [4] [14]. Furthermore, GAs are known for their inefficiency [15] [16] and tendency to identify unstable solutions—termed *lucky solutions*—which are situated on narrow peaks in the optimisation landscape and lack reliability in stochastic environments [17] [18] [19].

To overcome the limitations of GAs in MAP-Elites, specifically their inefficiency and inadequacy for evolving high-dimensional neural networks, significant advancements have been made by integrating Deep Reinforcement Learning (DRL) [20] [21] into the MAP-Elites framework. Unlike traditional QD optimisation, which aims to find a diverse set of solutions, DRL focuses on identifying a single, optimal solution through continuous interaction with the environment. This approach is enhanced by sophisticated gradient-based optimisation methods, enabled by advancements in deep learning. We see its application in the MAP-Elites framework where traditional GA-based variation operators are enhanced with Policy Gradient (PG) [22] [23] methods. These methods, a specialised branch of DRL, are particularly effective for training large neural networks to navigate high-dimensional state spaces and manage continuous action spaces [24] [25] [26]. Their integration into MAP-Elites leverages these capabilities, rendering the combined framework more capable of tackling complex optimisation challenges. For example, PGA-MAP-Elites [27] uses a PG variation operator to efficiently improve fitness (performance), while still using a GA variation operator to maintain diversity. Its PG operator relies on an actor-critic training, where the critic which is continuously updated during the learning process, guides the optimisation of solutions within the MAP-Elites' archive. DCG-MAP-Elites-AI [28] [29] is an extension of PGA-MAP-Elites that tries to promote more diversity in tasks where PGA-MAP-Elites struggled to, by conditioning the actor-critic training on the behavioural descriptors of the solutions. The synergy between MAP-Elites and gradient based methods, particularly through the two aforementioned actor-critic based algorithms, has proven powerful, efficiently generating a diverse set of high-performing solutions for specific problems. However, these algorithms are relatively slow due to the substantial time required for actor-critic training. Furthermore, these algorithms do not necessarily benefit from parallelisation because the actor-critic training requires a significant number of sequential learning steps to converge. This sequential process inherently restricts the scalability of these algorithms.

Exploring further into scalability, recent advancements in computational libraries have opened new avenues for achieving massive parallelisation, enhancing the scalability of algorithms. Libraries like EvoJax [30], QDax [31], and EvoSax [32] have significantly accelerated run-times by leveraging modern hardware accelerators such as GPUs. These devices enable the vectorisation of many computational operations, allowing for high levels of parallel evaluation. Moreover, the development of fast, highly-parallel simulators supports the application of these advances across a wide range of simple and complex tasks. Research by Lim et al. [33] shows that QD algorithms, particularly MAP-Elites, are highly scalable with parallel evaluations. By increasing the batch size per generation from around 100 to about 100,000, they were able to reduce the runtime of MAP-Elites by roughly 100 times without compromising performance, all using a single consumer-grade GPU. MEMES [34] has leveraged these advancements to improve another MAP-Elites based algorithm, ME-ES [35], which replaces the GA variation operator with an Evolution Strategies (ES) [36] algorithm. Instead of applying the ES algorithm sequentially to each solution, MEMES operates multiple independent ES workers in parallel, each with a distinct objective tailored for QD optimisation. These workers optimise their objectives by sampling and evaluating perturbations around a current solution to find empirical gradient estimates. Unlike the actor-critic based MAP-Elites algorithms, solutions in MEMES are optimised independently, without relying on any global training process. Therefore, it inherently possesses strong scalability capabilities, as its performance can always be enhanced by optimising more solutions in parallel. However, this method is sample inefficient because it requires running full episodes to evaluate solutions, and accurate gradient estimates necessitate averaging over many episodes.

Moreover, while MEMES has the potential to scale effectively, it demands substantial memory to fully benefit from parallelisation. Consequently, consumer-grade hardware may not be sufficient to achieve the desired benefits.

The goal of this project is to synergise PG methods with MAP-Elites to develop an algorithm that combines the fast execution and scalability of MAP-Elites with the efficient optimisation capabilities of PG methods. Specifically, we aim to achieve performance comparable to actor-critic based MAP-Elites algorithms but with significantly reduced runtime. Furthermore, we intend for the algorithm to remain scalable, meaning that additional hardware resources can enhance final performance or reduce training time without compromising final performance.

We therefore introduce the Monte Carlo Policy Gradient MAP-Elites (MCPG-ME) algorithm, which utilises a Monte Carlo Policy Gradient (MCPG) method in an off-policy fashion. This approach optimises solutions sample-efficiently while maintaining independent optimisation processes without relying on any global training process, thereby overcoming some limitations of actor-critic based algorithms. We compare our algorithm against the state-of-the-art actor-critic based MAP-Elites algorithms mentioned previously, as well as MAP-Elites and MEMES, across five and three continuous control locomotion tasks with episode lengths of 250 and 1000 timesteps, respectively. Given a fixed number of evaluations, our method: **(1)** achieves equal or higher diversity score (coverage) than all baselines across all tasks, **(2)** scores a higher QD score than both non-actor-critic based algorithms on all tasks, **(3)** outperforms PGA-MAP-Elites and DCG-MAP-Elites-AI in the majority and minority of the tasks, respectively, **(4)** trains significantly faster than actor-critic based algorithms (up to nine times faster), **(5)** reaches a competitive predetermined QD score faster than all baselines in most tasks, and **(6)** with adequate hardware support for parallelisation, increasing the number of solutions optimised in parallel does not compromise performance and simultaneously reduces runtime.

Chapter 2

Background

2.1 Reinforcement Learning

In this section, we delve into the foundational concepts of Reinforcement Learning (RL), exploring its core framework and briefly covering the Markov Decision Process (MDP), which provides the mathematical basis for modelling RL tasks. Additionally, we discuss various methods used in RL, including Monte Carlo (MC) and Policy Gradient (PG) methods, illustrating their mechanisms and roles in optimising policies. We introduce the REINFORCE algorithm, which serves as the base of our work, and outline the base of actor-critic methods, highlighting their strategies for reducing variance and improving learning efficiency. For simplicity, the mathematical explanations assume a discrete time space; however, these concepts can be easily applied to a continuous time space with only a few adjustments. Additionally, the theory used in this section is based on the foundational work presented in Sutton and Barto's RL book [37].

2.1.1 Summary of Notation

Capital letters are used for random variables, and lower case letters are used for the values of random variables and for scalar functions. Quantities which are real-valued vectors are written in bold and in lower case (the notation is adapted from [37]).

- s, s' state, next state (respectively)

- a action

- $\mathcal{S}$ set of all nonterminal states

- $\mathcal{A}(s)$ set of actions possible in state s

- $\mathcal{R}$ set of possible rewards

- t discrete time step

- T final time step of an episode

- S_t state at t

- A_t action at t

- R_t reward at t, dependent, like S_t, on A_{t-1} and S_{t-1}

- G_t return (cumulative discounted reward) following t

- π policy, decision-making rule

- $\pi(s)$ action taken in state s under deterministic policy π

- $\pi(a|s)$ probability of taking action a in state s under stochastic policy π

- $p(s', r|s, a)$ probability of transitioning to state s', with reward r, from s, a

- $v_\pi(s)$ value of state s under policy π (expected return)

- $q_\pi(s, a)$ value of taking action a in state s under policy π

- $V_t(s)$ estimate (a random variable) of $v_\pi(s)$

- $Q_t(s, a)$ estimate (a random variable) of $q_\pi(s, a)$

- $\hat{v}(s, w)$ approximate value of state s given a vector of weights w

- $\hat{q}(s, a, w)$ approximate value of state–action pair s, a given weights w

- w, w_t vector of (possibly learned) weights underlying an approximate value function

- γ discount-rate parameter

- ϵ probability of random action in ϵ-greedy policy

- α, β step-size parameters

- $Pr\{X = x\}$ probability that a random variable X takes on the value x

2.1.2 Framework

The Reinforcement Learning (RL) problem involves learning through interaction to accomplish a specific goal. The learner, also known as the decision-maker, is referred to as the *agent*, while the *environment* encompasses everything external the agent interacts with. This interaction is continuous: the agent chooses actions, and the environment responds to these actions by presenting new scenarios to the agent. Additionally, the environment provides rewards—specific numerical values—that the agent aims to maximize over time. A fully detailed description of an environment outlines a *task*, which is essentially one instance of the RL problem.

Formally, the agent and environment interact at a series of discrete time steps, denoted as $t = 0, 1, 2, \ldots$. At each time step t, the agent observes a specific representation of the environment's state, $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of all possible states. Based on this state, the agent selects an action, $A_t \in \mathcal{A}(S_t)$, where $\mathcal{A}(S_t)$ is the set of actions available in state S_t. Following this action, the agent transitions to a new state, S_{t+1}, and receives a numerical *reward*, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$.

At each time step, the agent executes a function that maps states to the probabilities of choosing each possible action. This function is known as the agent's *policy*, represented by π_t, where $\pi_t(a|s)$ indicates the conditional probability of selecting action $A_t = a$ when in state $S_t = s$. Essentially, reinforcement learning (RL) techniques determine how the agent modifies its policy based on its experiences. The ultimate aim for the agent is often to maximise the total rewards it accumulates over time.

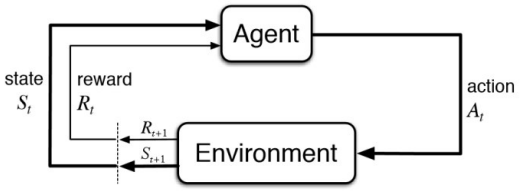


Figure 2.1: The interaction loop between the agent and the environment in reinforcement learning.

2.1.3 Markov Decision Process

In simple words, the Markov property states that the future is independent of the past given the present. More formally, in the context of RL, the environment and task as a whole have the Markov property if and only if:

$$Pr\{R_{t+1} = r, S_{t+1} = s' \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\} \tag{2.1}$$

is equal to:

$$p(s', r \mid s, a) = Pr\{R_{t+1} = r, S_{t+1} = s' \mid S_t, A_t\} \tag{2.2}$$

for all s', r, and histories $S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t$. If an RL task satisfies the Markov property, it is called a Markov decision process (MDP).

In this work, we assume that the RL tasks can be modelled as an MDP.

2.1.4 Important Concepts

After establishing the framework of RL and MDPs, it is essential to understand some general concepts in RL that apply across different methods.

Discounting Factor

The *discount factor*, denoted as γ, is a parameter in the range $0 \leq \gamma \leq 1$ that determines the present value of future rewards. It ensures that immediate rewards are given more weight than distant future rewards, addressing the problem of infinite returns in ongoing tasks. The discounted return (cumulative reward) G_t at time step t is defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + .. \tag{2.3}$$

We will refer to it as *reward-to-go*.

Trajectories and episodes

Trajectory is a sequence of states, actions, and rewards that the agent experiences as it interacts with the environment. It is denoted as:

$$\tau = (S_0, A_0, R_1, S_1, A_1, R_2, ..), \tag{2.4}$$

Episode is a finite trajectory that begins in an initial state and ends in a terminal state. Each episode represents a complete sequence of interactions from start to finish, allowing the agent to reset and start a new sequence.

State-Value and Action-Value Functions

A *state-value* function,$v_\pi(s)$, represents the expected return when starting from state s and following policy π. Formally, it is defined as:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right] \tag{2.5}$$

An *action-value* function, $q_\pi(s, a)$, represents the expected return when starting from state s, taking action a, and thereafter following policy π. Formally, it is defined as:

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] \tag{2.6}$$

Exploration vs. Exploitation

A fundamental challenge in RL algorithms is balancing exploration and exploitation. *Exploration* involves trying new actions to discover their effects and improve its knowledge of the environment, which is crucial for ensuring that the agent does not miss out on potentially better actions. Exploration can be categorised into '*parameter-based*' exploration, where randomness is introduced into the parameter space of the policy itself, and '*action-based*' exploration, where noise is injected on the action space, and involves selecting actions directly based on their estimated probability to encourage trying less frequent actions. In contrast, *exploitation* involves the agent selecting the best-known action based on its current knowledge to maximise the reward, making the most of the agent's current understanding to achieve its goal. A common approach to balance exploration and exploitation is to use ϵ-greedy policies, where the agent predominantly exploits but occasionally explores (in the action space) with a small probability ϵ. Mathematically, the action selection in an ϵ-greedy policy can be described by the following probability distribution:

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|} & \text{if } a = \arg\max_a q(s, a) \\ \frac{\epsilon}{|\mathcal{A}(s)|} & \text{otherwise} \end{cases} \tag{2.7}$$

where $|\mathcal{A}(s)|$ is the number of possible actions in state s.

On-Policy vs. Off-Policy Methods

On-policy methods refer to approaches where the policy being evaluated and improved is the same as the policy used to generate the behaviour. In other words, the agent learns the value of the policy that is currently following. The primary advantage of these methods is their simplicity and directness in improving the policy being followed. However, they can suffer from inefficiency due to the necessity of continually balancing exploration and exploitation within the same policy framework. Additionally, the convergence of on-policy methods can be slow, particularly in environments with large state-action space.

Off-policy methods, in contrast, involve learning the value of one policy—the *target policy*—while using a different policy—the *behaviour policy*—to generate behaviour. This separation allows greater flexibility, as the behaviour policy can be designed to explore the environment more thoroughly while the target policy focuses on optimal performance. This framework, facilitates

a more efficient exploration and faster convergence compared to on-policy methods, typically making it more sample-efficient.

2.1.5 Policy Gradient Methods

In RL, most traditional methods have relied heavily on action-value functions to determine optimal policies. These methods estimate the value of actions and select actions based on these estimated values. Unlike action-value methods, Policy Gradient (PG) methods directly parameterise the policy and select actions based on this parameterised policy without necessarily consulting a value function.

We denote the policy's parameter vector as $\boldsymbol{\theta} \in \mathbb{R}^d$. Therefore, we write $\pi(a|s, \boldsymbol{\theta}) = Pr(A_t = a|S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta})$ to represent the probability of taking action a at time t given the state s at time t and parameter $\boldsymbol{\theta}$. PG methods aim to optimise a scalar performance measure $J(\boldsymbol{\theta})$—find the parameter $\boldsymbol{\theta}^*$ which maximises J–by adjusting the policy parameter in the direction of the performance gradient. These methods use gradient ascent to maximise performance, updating the parameter as follows:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\widehat{\nabla J(\boldsymbol{\theta}_t)} \tag{2.8}$$

where $\widehat{\nabla J(\boldsymbol{\theta}_t)}$ is a stochastic estimate approximating the gradient of the performance measure with respect to $\boldsymbol{\theta}_t$. This approach applies to all methods that follow this general schema, regardless of whether they also learn an approximate value function. In that case, when methods learn approximations for both policy and value functions, they are often referred to as actor-critic methods, with the "*actor*" representing the policy, and the "*critic*" representing the value function. For the following explanations, the episodic case is considered, as it is the one relevant to our work, where the performance is defined as the value of the start state under the parameterised policy. Furthermore, for simplicity purposes and without losing any meaningful generality, we assume that every episode starts in some non-random state, s_0, and that there is no discounting factor ($\gamma = 1$) for the episodic case. Then, formally, the performance is defined as:

$$J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0) = \mathbb{E}(G_0|S_0 = s_0) \tag{2.9}$$

Adjusting the policy parameters to ensure performance improvement can be challenging due to the dependency of performance on both action selections and the distribution of states in which those selections are made, both influenced by the policy parameter. Although the effect of the policy parameter on action and rewards can be computed straightforwardly, the effect on state distribution is typically unknown. Notably, the policy gradient theorem provides an analytic expression for the gradient of performance with respect to the policy parameter, bypassing the need for the derivative of the state distribution. The policy gradient theorem for the episodic case states:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s, a)\nabla\pi(a|s, \boldsymbol{\theta}), \tag{2.10}$$

where the gradients are column vectors of partial derivatives with respect to the components of $\boldsymbol{\theta}$, and π denotes the policy corresponding to parameter vector $\boldsymbol{\theta}$. In the episodic case, the constant of proportionality is the average length of an episode. Here, $\mu(s)$ represents the stationary distribution over the states under the policy π, indicating how often the agent visits each state in the long run while following the policy. Note that if we did not assume that every episode starts at the same state, the performance measure would be defined as the expected value over all possible start states, weighted by their probability of being the initial state, and the policy gradient theorem would accordingly incorporate the initial state distribution.

2.1.6 Monte Carlo Methods

Monte Carlo (MC) methods constitute a broad class of computational algorithms that leverage repeated random sampling to obtain numerical results. The fundamental principle underlying these methods is the utilisation of randomness to address problems that are inherently deterministic. These methods involve generating random samples from a specified probability distribution, which are then used to approximate solutions to mathematical problems that may be analytically intractable. Apart from their simplicity and flexibility, one of their main advantages, which is of particular interest in this work, is that they are well-suited to parallel computing since each sample can be processed independently.

In the context of RL, MC methods are employed to estimate the state-value functions and action-value functions. By averaging the returns obtained from multiple episodes, MC methods provide unbiased estimates of these functions. It is important to emphasise that these methods are episodic, meaning they rely on complete episodes, from the initial state to a terminal state, to update the value functions. This dependency on complete episodes can be a significant limitation in environments where episodes are lengthy or reaching a terminal state is rare, as it delays policy updates, leading to sample inefficiency. Moreover, despite providing unbiased estimates, a major drawback of MC methods is their tendency to exhibit high variance in these estimates. This high variance can result in unstable learning, requiring a large number of episodes to achieve stable and accurate value estimates, thus further diminishing the methods' efficiency.

To further explore the bias and variance of the estimates produced by the MC methods in the context of PG methods, let's continue with the same notation and assumptions as in the previous section. Consider the performance function, defined as the expected return from a fixed starting state, s_0.

$$J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0) = \mathbb{E}_{\pi}\left[G_0 | S_0 = s_0\right] \tag{2.11}$$

Let X be the random variable $G_0|S_0 = s_0$, and suppose, $X \sim F(\mu, \sigma^2)$. Moreover suppose we sample N empirical returns from the same distribution that X follows, and that each empirical return is independent of each other. That is, let $X_i \sim F(\mu, \sigma)$ where X_i represents the empirical return of the i^{th} episode run by the agent. Then, we estimate $\mathbb{E}[X]$ to be $\bar{X}$, where $\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$.

$\bar{X}$ is an unbiased estimator of $\mathbb{E}[X]$ since,

$$\mathbb{E}[\bar{X}] = \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} X_i\right] \tag{2.12}$$

$$= \frac{1}{N}\mathbb{E}\left[\sum_{i=1}^{N} X_i\right] \quad \text{(linearity of expectation)} \tag{2.13}$$

$$= \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}[X_i] \tag{2.14}$$

$$= \frac{1}{N} \times N\mu = \mu = \mathbb{E}[X]. \tag{2.15}$$

However, the variance of the estimator is,

$$\text{Var}(\bar{X}) = \text{Var}\left(\frac{1}{N}\sum_{i=1}^{N} X_i\right) \tag{2.16}$$

$$= \frac{1}{N^2}\text{Var}\left(\sum_{i=1}^{N} X_i\right) \quad \text{(since } X_i \text{ are independent)} \tag{2.17}$$

$$= \frac{1}{N^2}\sum_{i=1}^{N}\text{Var}(X_i) \tag{2.18}$$

$$= \frac{1}{N^2} \times N\sigma^2 = \frac{\sigma^2}{N}. \tag{2.19}$$

Therefore, as discussed, MC methods inherently exhibit a trade-off between sample efficiency and the variance of their estimates of the value functions.

While the reduction of variance through the averaging of multiple independent samples is beneficial, in off-policy learning, where policies used to generate behaviour differ from the target policy being improved, a further adjustment is necessary. A technique of high interest in this work, mainly used in off-policy Monte Carlo methods, is *importance sampling* (IS). This technique adjusts the value estimates by a factor that accounts for the difference between the behaviour policy and the target policy. This adjustment, known as the *importance sampling ratio*, corrects for the bias introduced by the discrepancy between the two policies. In particular, suppose that we want to estimate

$$\mathbb{E}_{P(X)}[f(X)] = \int_x P(x)f(x)dx \tag{2.20}$$

where f and P is a general function and a probability measure respectively. Furthermore, suppose that it is difficult to obtain samples from P, but we can sample from a different distribution, say Q. In that case, given samples from Q, we can estimate the expectation with respect to P as:

$$\mathbb{E}_{P(X)}[f(X)] = \int_x P(x)f(x)dx \tag{2.21}$$

$$= \int_x \frac{Q(x)}{Q(x)}P(x)f(x)dx \tag{2.22}$$

$$= \int_x Q(x)\left[\frac{P(x)}{Q(x)}f(x)\right]dx \tag{2.23}$$

$$= \mathbb{E}_{Q(X)}\left[\frac{P(x)}{Q(x)}f(X)\right] \tag{2.24}$$

$$\approx \frac{1}{m}\sum_{i=1}^{m}\frac{P(x^{(i)})}{Q(x^{(i)})}f(x^{(i)}) \quad \text{with } x^{(i)} \sim Q \tag{2.25}$$

In the above, we assume $Q(x) = 0 \Rightarrow P(x) = 0$. Hence, one can sample from a different distribution Q and then simply re-weight the samples to obtain an unbiased estimate [38].

2.1.7 REINFORCE

A direct implementation of the PG theorem using MC methods is the REINFORCE algorithm (see Algorithm 1). REINFORCE, a Monte Carlo policy gradient (MCPG) method, updates the policy parameters based on the returns from complete episodes. It directly estimates the gradient of the expected return with respect to the policy parameters and adjusts these parameters in a

direction that increases the expected return.

To better understand the underlying mechanics of REINFORCE, recall the overall strategy of stochastic gradient ascent, which requires obtaining samples whose expected gradient is proportional to the actual gradient of the performance measure with respect to the parameters. The sample gradients only need to be proportional to the gradient because any constant of proportionality can be absorbed into the step size α, which is otherwise arbitrary. The PG theorem provides an exact expression proportional to the gradient. The primary task is to devise a sampling method whose expected value approximates this formula. Observe that the right-hand side of the PG theorem comprises a summation over states, weighted by their frequency under the target policy π. Therefore, when following the policy π, states will naturally occur in these specified proportions, thus:

$$\begin{aligned}\nabla J(\boldsymbol{\theta}) &\propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta}) \\ &= \mathbb{E}_\pi \left[\sum_a q_\pi(S_t,a) \nabla \pi(a|S_t,\boldsymbol{\theta}) \right]. \end{aligned} \tag{2.26}$$

From here, we could instantiate the stochastic gradient-ascent algorithm as:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \sum_a \hat{q}(S_t,a,\mathbf{w}) \nabla \pi(a|S_t,\boldsymbol{\theta}), \tag{2.27}$$

where $\hat{q}$ is some learned approximation to q^π.

The derivation of REINFORCE progresses by incorporating A_t. Although Equation (2.26) includes a summation over actions, each term lacks the necessary weighting by $\pi(a|S_t,\boldsymbol{\theta})$ for an expectation under π. To maintain the equation's integrity while introducing this weighting, we multiply the summed terms by $1 = \frac{\pi(a|S_t,\boldsymbol{\theta})}{\pi(a|S_t,\boldsymbol{\theta})}$, effectively adding the required weighting without changing the equality.

$$\begin{aligned}\nabla J(\boldsymbol{\theta}) &\propto \mathbb{E}_\pi \left[\sum_a \pi(a|S_t,\boldsymbol{\theta}) q_\pi(S_t,a) \frac{\nabla \pi(a|S_t,\boldsymbol{\theta})}{\pi(a|S_t,\boldsymbol{\theta})} \right] \\ &= \mathbb{E}_\pi \left[q_\pi(S_t,A_t) \frac{\nabla \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})} \right] && \text{(replacing } a \text{ by the sample } A_t \sim \pi) \\ &= \mathbb{E}_\pi \left[G_t \frac{\nabla \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})} \right], && \text{(because } \mathbb{E}_\pi[G_t|S_t,A_t] = q_\pi(S_t,A_t)) \end{aligned}$$

This expression, which can be sampled at each time step, is proportional to the gradient. Using this sample in the stochastic gradient ascent algorithm yields the REINFORCE update:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}.$$

The REINFORCE update can be understood intuitively: each update step is proportional to the reward-to-go G_t multiplied by a specific vector—the gradient of the logarithm of the action's probability. This vector indicates the direction in parameter space that would most increase the likelihood of selecting action A_t again when revisiting state S_t. The parameters are then adjusted in this direction, scaled by the return and inversely scaled by the probability of the action. This method ensures that actions leading to higher returns are more likely to be chosen, while preventing overly frequent actions from being disproportionately favoured.

In summary, REINFORCE uses the complete return from time t—reward-to-go—including all future rewards until the end of the episode, making it an MC algorithm. It is well-defined only for the episodic case, with policy updates occurring only after each episode concludes, aligning with traditional MC approaches discussed previously. The expected update direction over an

Algorithm 1 REINFORCE. Adapted from [37].

Input: a differentiable policy parameterisation $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialise policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$**:**
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$ ▷ $\nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}) = \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}$; follows from the identity $\nabla \ln x = \frac{\nabla x}{x}$

episode aligns with the performance gradient, assuring improvement in expected performance for a sufficiently small α, and ensuring convergence to a local optimum under typical stochastic approximation conditions when α decreases. However, being an MC method, REINFORCE is subject to high variance in its updates, which can lead to instability and slow convergence in the learning process.

A technique that keeps the gradient estimate unbiased, and significantly reduces its variance is to use a *baseline*, $b(s)$. The policy gradient theorem can be generalised to include a comparison of the action value to an arbitrary baseline $b(s)$:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \left(q_\pi(s, a) - b(s)\right) \nabla \pi(a|s, \boldsymbol{\theta}). \tag{2.28}$$

For the baseline to be unbiased, it should be any function or random variable that does not vary with action a. To see this, suppose $b(s)$ is independent to a, then:

$$\sum_a b(s) \nabla \pi(a|s, \boldsymbol{\theta}) = b(s) \nabla \sum_a \pi(a|s, \boldsymbol{\theta}) = b(s) \nabla 1 = 0,$$

therefore the subtracted term in the summation over the action space in (2.28) is zero, and the expectation shown in (2.26) gets subtracted by $\mathbb{E}_\pi(0) = 0$, which means it remains unbiased.

A logical option for the baseline is to use an estimate of the state value, $\hat{v}(S_t, \boldsymbol{w})$, where $\boldsymbol{w} \in \mathbb{R}^d$ represents learned weight vector. This estimated state-value function calculates the value of the initial state in each state transition. This value provides a reference point for the return that follows, but since it is determined before the action of the transition, it cannot be utilised to evaluate the specific action taken.

Conversely, the state-value function can also applied to the next state in the transition. The estimated value of this next state, when discounted and added to the reward, forms the one-step return, $G_{t:t+1}$, which serves as a useful estimate of the actual return and provides means to evaluate the action. Although this one-step return introduces bias, it has been shown that its difference with the current value-state function—*temporal difference*—is often superior to the actual return in terms of its variance [39]. When the state-value function is employed to assess actions like this, it is referred to as a *critic*. Consequently, the overall policy-gradient approach

in which it is used is known as an *actor-critic* method. See below the corresponding update rule:

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &\doteq \boldsymbol{\theta}_t + \alpha \left(G_{t:t+1} - \hat{v}(S_t, \mathbf{w})\right) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)} \\ &= \boldsymbol{\theta}_t + \alpha \left(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})\right) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)} \\ &= \boldsymbol{\theta}_t + \alpha \delta_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}.\end{aligned}$$

2.1.8 Integrating Deep Learning with Reinforcement Learning

The integration of deep learning (DL) with RL, forming what is known as Deep Reinforcement Learning (DRL), addresses several limitations inherent in traditional RL methods. Deep learning models, particularly deep neural networks (DNNs), are adept at handling high-dimensional, non-linear problems. By parameterising the policy $\pi(a|s, \boldsymbol{\theta})$ or the value function with DNNs, DRL methods can operate effectively in environments with complex high dimensional state representations and continuous action spaces that are challenging for traditional methods.

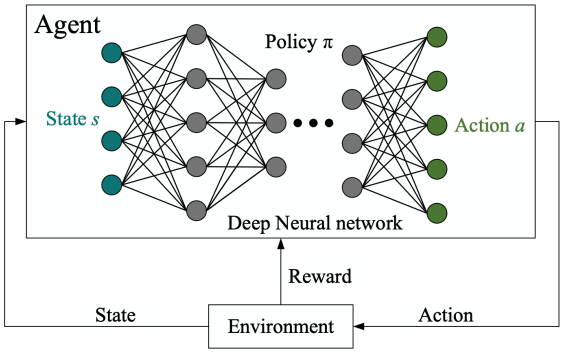


Figure 2.2: The interaction loop between the agent and the environment in deep reinforcement learning. Adapted from [40].

2.2 Evolutionary Algorithms

In this section, we explore Evolutionary Algorithms (EAs), a class of optimisation algorithms well-suited for black-box functions where traditional optimisation techniques, such as gradient descent, are ineffective. We will discuss the foundational concepts of EAs, their operators, and delve into specific algorithms such as Quality-Diversity (QD) algorithms and Evolution Strategies (ES). Special focus is given to the MAP-Elites algorithm, a prominent QD algorithm that is the main component of this work.

Traditional optimisation techniques, such as stochastic gradient descent, are highly effective for optimising non black-box functions [41]. In these cases, the internal structure of the function is known and accessible, allowing for the calculation of gradients. Gradient descent, for instance, uses the gradient of the objective function to iteratively adjust the parameters in the direction that reduces the function's value, efficiently finding local minima. These techniques rely on smoothness of the function, or a surrogate of the function to navigate towards optimal solutions. However, for black-box functions, where the internal workings and gradients are not available, and only the inputs and outputs can be observed, traditional methods like gradient descent are ineffective. Instead, back-box optimisation process must rely solely on the evaluation of these inputs and outputs to guide the search for optimal solutions.

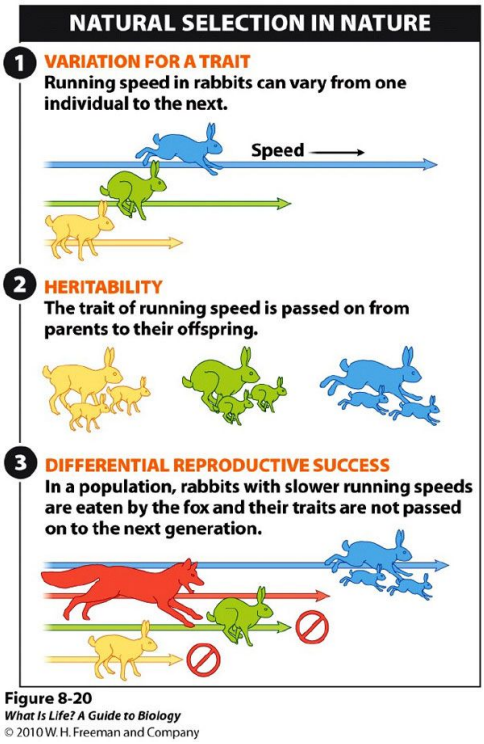


Figure 2.3: Natural Selection in Nature. Adapted from [43].

Evolutionary Algorithms (EAs) is a family of optimisation algorithms for black-box functions, inspired by natural selection and concepts from genetics. The theory of natural selection was introduced in Charles Darwin's book *On the Origin of Species* in 1859 [42], mainly relying on four key concepts (see Figure 2.3):

1. **Variation**: Individuals within the same species exhibit variations, and many of these variations are passed down from parents to offspring.

2. **Resource Limitation**: Resources such as food or shelter are limited, meaning that not all individuals can survive and reproduce.

3. **Survival of the Fittest**: Individuals with heritable traits better suited to the environment are more likely to survive and reproduce.

4. **Inheritance of Advantageous Traits**: The individuals that survive and reproduce pass their heritable advantageous traits to their offspring, increasing the frequency of these traits in the population over generations.

Building on Darwin's theory, Gregor Mendel's work in the middle 19^{th} century on inheritance patterns in pea plants [44] laid the foundation for understanding genetic inheritance, known as Mendelian genetics. Mendel discovered that traits are inherited discretely through units called genes, which follow specific patterns of dominance and segregation. This understanding provided the mechanism for how traits are passed from one generation to the next, complementing Darwin's theory by explaining the source of heritable variations.

The integration of Darwinian natural selection and Mendelian genetics led to the development of Neo-Darwinism, or the modern synthesis, in the early 20th century. This framework combines

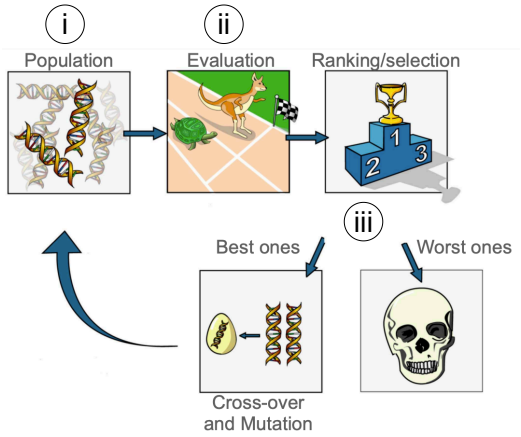


Figure 2.4: Main concept of EAs. Adapted from [45].

the concepts of mutation, genetic variation, and natural selection to explain the adaptive evolution of populations. Evolutionary Algorithms (EAs) are based on Neo-Darwinism, and their common base is the following (see Figure 2.4):

i. They manipulate a population of solutions, initially randomly generated, where each solution is represented by a *genotype*—a set of genes.

ii. The genotype can be developed into a *phenotype*—the physiological expression of the genotype—which is then evaluated using a fitness function to measure performance.

iii. The fittest individuals have a higher probability of passing part of their genotype to their offspring, where an offspring refers to the generated genotype when the parent genotype is varied, ensuring that advantageous traits are propagated through successive generations.

2.2.1 Quality-Diversity Algorithms

Quality-Diversity (QD) algorithms represent a family of evolutionary algorithms (EAs) that prioritise both high-quality and diverse solutions [5]. Quality refers to the fitness of a solution—how well it addresses the specific problem, as determined by a problem-specific fitness function. Diversity, on the other hand, is assessed based on how the solutions differ from one another. This diversity is quantified using a set of features known as a behavioural descriptor (BD), which helps in forming a behavioural characterisation of solutions. These descriptors define a BD space, the domain over which the diversity of solutions is measured.

Unlike traditional optimisation algorithms that focus solely on identifying the highest-performing solutions, QD algorithms diverge by aiming to generate a large collection of unique and effective solutions that maximize both performance and diversity across the BD space (see Figure 2.5). This approach allows for a broader exploration of the problem space, retaining a spectrum of solutions rather than just the optimal ones. The overall performance of a QD algorithm is evaluated by the quality of the produced collection of solutions according to three criteria: coverage of the BD space—the percentage of the descriptor space covered by the solutions; the uniformity of this coverage; and the performance of the solution found for each type [4].

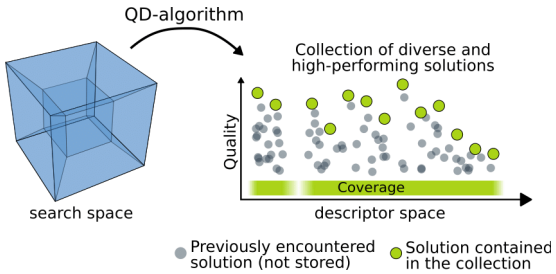


Figure 2.5: The objective of a QD-algorithm is to generate a collection of both diverse and high-performing solutions. This collection represents a (model free) projection of the highdimensional search space into a lower dimensional space defined by the solution descriptors. The quality of a collection is defined by its coverage of the BD space and by the global quality of the solutions that are kept in the collection. Adapted from [4].

Most QD algorithms follow a similar sequence of steps that are repeated until termination criterion is reached: a container gathers all the solutions found so far into an ordered collection, where only the best (high-performing) and most diverse solutions are kept. Based on a task- or algorithm-specific selection operator, a subset of the solutions in the container is selected for random variation. The selected and mutated solutions are then evaluated using the aforementioned measures to determine their fitness value and BD. Finally, based on algorithm-specific criteria, these new solutions are either added to the container, replace older solutions, or are completely ignored.

2.2.1.1 MAP-Elites

MAP-Elites [14] is a simple, yet effective QD optimisation algorithm that has shown success to a variety of fields. It has been used to teach robots how to adapt to damage [9] [10] [11], generate aerodynamic designs [46] or to create content for games [47] [48]. MAP-Elites is one of the most popular QD algorithms, managing to find a wide set of high-performing solutions, while being conceptually simple and easy to implement. First, a user chooses a fitness function $f(x)$ that evaluates a solution x. For instance, if searching for robot locomotions, the fitness function could be how fast the robot could move from one specified location to another. Second, the user chooses N dimensions of variation of interest that define a feature space of interest to the user—the BD space mentioned previously. For robot locomotions, one dimension of interests could be the time duration a specific foot is in touch with the ground in one episode. Similarly, other dimensions could represent the time duration corresponding to the other feet of the robot.

Each dimension of variation is discretised based on user preference or available computational resources, and given a particular discretisation, MAP-Elites will search for the fittest solution for each cell in the N-dimensional BD space. This search is conducted in the space of all possible values of x—the search space—where x is the description vector of a candidate solution. In the EAs vocabulary, the descriptor x is called a genotype, and the robot locomotion, in our example, is the phenotype, p_x. Moreover, a behaviour function, $b(x)$ must exist in order to map the genotype to an N-dimensional vector, $\mathbf{b_x}$, which describes x's features.

MAP-Elites begins by randomly generating $K \in \mathbb{N}$ genotypes and assessing each for fitness (performance) and features (behaviour). These genotypes are then assigned to corresponding cells in the behavioural space. If multiple genotypes map to the same cell, only the fittest one is retained. Once initialised, the algorithm proceeds with the following steps, repeated until a termination criterion is met (see Figure 2.6 & Algorithm 2):

i. **Selection Mechanism**: A genotype is randomly selected from one of the cells in the map.

ii. **Emission Stage**: The selected genotype produces an offspring through mutation—introducing random changes to the genotype (injecting noise into the parameters)—and/or crossover, which combines the genotypes of two parents.

iii. **Evaluation**: The offspring's behaviour and fitness are evaluated.

iv. **Addition Mechanism**: The offspring is assigned to a cell based on its behavioural characteristics. If the cell is empty, the offspring is placed directly. If the cell is occupied, the offspring replaces the current occupant only if it has higher fitness; otherwise, the original occupant is retained and the offspring is discarded, or '*dies*'

It is important to emphasise that a multitude of genotypes, often infinite, can map to the same cell within the behaviour space. The complex many-to-one mapping from a genotype x to its behaviour vector $\mathbf{b_x}$ makes it impossible to directly search within the behaviour space, as it is uncertain which genotype will map to specific cells. This uncertainty is a key reason why MAP-Elites is valuable—it essentially attempts an exhaustive search in the behaviour space, aiming to identify the highest-performing solution for each cell.

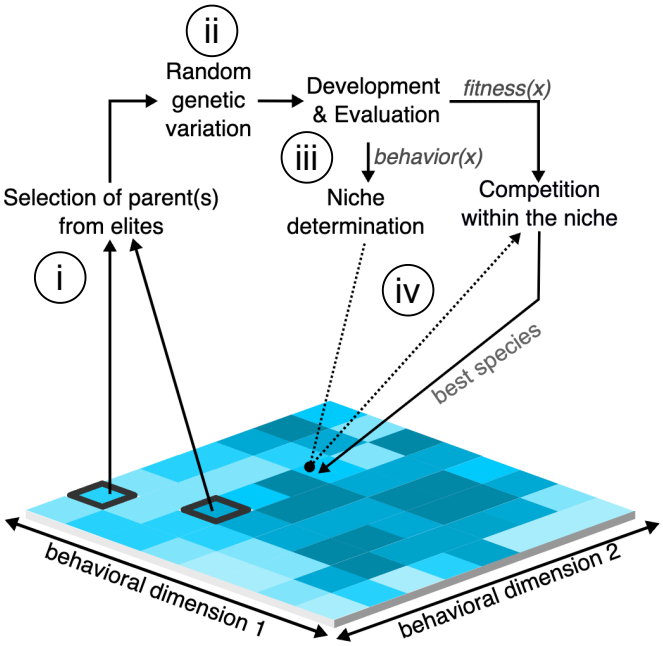


Figure 2.6: Main cycle of MAP-ELites. Adapted from [49].

Algorithm 2 MAP-Elites Algorithm. Adapted from [14].

procedure MAP-ELITES ALGORITHM
  $\mathcal{P} \leftarrow \emptyset, \mathcal{X} \leftarrow \emptyset$ ▷ Create an empty, N-dimensional map of elites: {solutions $\mathcal{X}$ and their performances $\mathcal{P}$}
  for iter $= 1 \rightarrow I$ **do** ▷ Repeat for I iterations
    if iter $< G$ **then** ▷ Initialize by generating G random solutions
      $\mathbf{x}' \leftarrow$ random_solution()
    else
      $\mathbf{x} \leftarrow$ random_selection($\mathcal{X}$) ▷ Randomly select an elite $\mathbf{x}$ from the map $\mathcal{X}$
      $\mathbf{x}' \leftarrow$ random_variation($\mathbf{x}$) ▷ Create $\mathbf{x}'$, a randomly modified copy of $\mathbf{x}$ (via mutation and/or crossover)
    end if
    $\mathbf{b}' \leftarrow$ feature_descriptor($\mathbf{x}'$) ▷ Simulate the candidate solution $\mathbf{x}'$ and record its feature descriptor $\mathbf{b}'$
    $\mathbf{p}' \leftarrow$ performance($\mathbf{x}'$) ▷ Record the performance $\mathbf{p}'$ of $\mathbf{x}'$
    if $\mathcal{P}(\mathbf{b}') = \emptyset$ or $\mathcal{P}(\mathbf{b}') < \mathbf{p}'$ **then** ▷ If the appropriate cell is empty or its occupants's performance is $\leq \mathbf{p}'$, then
      $\mathcal{P}(\mathbf{b}') \leftarrow \mathbf{p}'$ ▷ store the performance of $\mathbf{x}'$ in the map of elites according to its feature descriptor $\mathbf{b}'$
      $\mathcal{X}(\mathbf{b}') \leftarrow \mathbf{x}'$ ▷ store the solution $\mathbf{x}'$ in the map of elites according to its feature descriptor $\mathbf{b}'$
    end if
  end for
  return $\mathcal{P}$ and $\mathcal{X}$ ▷ return feature-performance map ($\mathcal{P}$ and $\mathcal{X}$)
end procedure

2.2.2 Evolution Strategies

Evolution Strategies (ES) belong to the category of EAs and operate in a similar cyclical process: in each iteration or "generation," a population of parameter vectors, known as "genotypes," is modified or "mutated," and their performance measure, or "fitness," is evaluated. The best-performing parameter vectors are then selected and combined to create a new population for the subsequent generation. This cycle is repeated until the objective is fully optimised or a stopping condition is met. The specific methods used to manage the population and execute mutations and recombinations vary among algorithms within this class.

2.2.2.1 Natural Evolution Strategies

A class of ES algorithms that is often used in synergy with MAP-Elites is the natural evolution strategies (NES) [50] [51]. Let F represent the objective function acting to parameters $\boldsymbol{\theta}$. NES algorithms model the population with a distribution over parameters $p_{\boldsymbol{\psi}}(\boldsymbol{\theta})$, where $\boldsymbol{\psi}$ are the parameters defining this distribution. The goal is to maximise the average objective value $\mathbb{E}_{\boldsymbol{\theta}\sim p_{\boldsymbol{\psi}}}[F(\boldsymbol{\theta})]$ across the population by optimising $\boldsymbol{\psi}$ through stochastic gradient ascent. In particular, using the score function estimator for $\nabla_{\boldsymbol{\psi}}\mathbb{E}_{\boldsymbol{\theta}\sim p_{\boldsymbol{\psi}}}[F(\boldsymbol{\theta})]$, in a manner similar to REINFORCE, NES algorithms take gradient steps on $\boldsymbol{\psi}$ with the following estimator:

$$\nabla_{\boldsymbol{\psi}}\mathbb{E}_{\boldsymbol{\theta}\sim p_{\boldsymbol{\psi}}}[F(\boldsymbol{\theta})] = \mathbb{E}_{\boldsymbol{\theta}\sim p_{\boldsymbol{\psi}}}\{F(\boldsymbol{\theta})\nabla_{\boldsymbol{\psi}}\log p_{\boldsymbol{\psi}}(\boldsymbol{\theta})\} \tag{2.29}$$

In this work, we concentrate on RL problems, therefore, let $F(\cdot)$ represent the stochastic return from an environment, and $\boldsymbol{\theta}$ denote the parameters of a policy $\pi_{\boldsymbol{\theta}}$, describing the actions of the agent in that environment. Then, the population distribution $p_{\boldsymbol{\psi}}$ is instantiated as an isotropic multivariate Gaussian with mean $\boldsymbol{\psi}$ and fixed covariance $\sigma^2\boldsymbol{I}$. This allows for the average objective, $\mathbb{E}_{\boldsymbol{\theta}\sim p_{\boldsymbol{\psi}}}[F(\boldsymbol{\theta})]$ to be expressed in terms of a mean parameter vector $\boldsymbol{\theta}$ directly in the following way:

$$\mathbb{E}_{\boldsymbol{\theta}\sim p_{\boldsymbol{\psi}}}[F(\boldsymbol{\theta})] = \mathbb{E}_{\boldsymbol{\epsilon}\sim N(0,\boldsymbol{I})}[F(\boldsymbol{\theta}+\sigma\boldsymbol{\epsilon})] \tag{2.30}$$

Hence, as now the re-parameterised version of the average objective function is solely expressed in terms of $\boldsymbol{\theta}$, it can be optimised over $\boldsymbol{\theta}$ directly using stochastic gradient ascent with the score function estimator:

$$\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\boldsymbol{\epsilon}\sim N(0,\boldsymbol{I})}[F(\boldsymbol{\theta}+\sigma\boldsymbol{\epsilon})] = \frac{1}{\sigma}\mathbb{E}_{\boldsymbol{\epsilon}\sim N(0,\boldsymbol{I})}\{F(\boldsymbol{\theta}+\sigma\boldsymbol{\epsilon})\boldsymbol{\epsilon}\} \tag{2.31}$$

which can be approximated with samples (see Algorithm 3)

Algorithm 3 Natural Evolution Strategies. Adapted from [51].

Input: Learning rate α, noise standard deviation σ, initial policy parameters $\boldsymbol{\theta}_0$
for $t = 0, 1, 2, \ldots$ **do**
  Sample $\boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_n \sim \mathcal{N}(0, \boldsymbol{I})$
  Compute returns $F_i = F(\boldsymbol{\theta}_t + \sigma\boldsymbol{\epsilon}_i)$ for $i = 1, \ldots, n$
  Set $\boldsymbol{U} \leftarrow \frac{1}{n\sigma}\sum_{i=1}^{n} F_i\boldsymbol{\epsilon}_i$ ▷ Gradient estimate approximated with samples
  Set $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha\boldsymbol{U}$
end for

2.3 Related Work

Quality-Diversity (QD) optimisation has emerged as a powerful paradigm in evolutionary computation, aiming to generate a diverse set of high-performing solutions. Among the various QD algorithms, MAP-Elites [14] stands out due to its simplicity and effectiveness in exploring a multidimensional behaviour space. The traditional MAP-Elites algorithm operates by emitting new solutions through random variations of existing high-performing solutions, typically using Genetic Algorithms (GAs)—randomly mutate and/or crossover parent genotypes to generate offspring. While effective in maintaining diversity, GAs can be inefficient [15] [16], especially for high-dimensional optimisation problems. To address these limitations, recent works have proposed the integration of directed search methods into the MAP-Elites framework. These methods replace or partially replace the GA emitter with more powerful search techniques, aiming to improve the efficiency and performance of the search process. These integrations can be broadly categorised into two families: gradient-based methods and evolutionary methods. In this section, we will discuss some of the MAP-Elites based algorithms from each category. Note that only the emitter (variation operator) component of the MAP-Elites algorithm will be discussed, as the rest of the cycle mostly follows the conventional MAP-Elites algorithm (2).

2.3.1 MAP-Elites with a Policy Gradient Based Emitter

The Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm, commonly used with the MAP-Elite integration, is a gradient-based method within the family of Deep Reinforcement Learning (DRL) algorithms, specifically those derived from policy gradient (PG) methods. TD3 is an actor-critic off-policy method specifically designed to address common pitfalls in PG methods, such as overestimation bias and high variance in value estimates [52]. It has been shown to be a robust and scalable approach for problems where the action space is continuous—*continuous control problems*—making it suitable for replacing GAs and integrating with MAP-Elites.

More specifically, the TD3 algorithm uses an actor-critic methodology to learn a deterministic policy (actor) indirectly via the maximisation of the action-value function. It calculates the performance gradient via deterministic policy gradient techniques and uses critics, a pair of deep neural networks, to approximate the action-value function using the Bellman equation. The updates of both the actor and the critics are performed using transitions from a replay buffer, which contains past experiences of the agent, making the algorithm off-policy due to the use of older versions of the policy. To avoid action-value overestimation and promote stability, TD3 incorporates several techniques such as using the minimum value from the two critics, employing target networks for both the actor and the critics, adding noise to the target actions, and updating the policy less frequently than the action-value function.

Policy Gradient Assisted MAP-Elites (PGA-MAP-Elites) [27] is one of the first state-of-the-art algorithms to use PG methods for the variation operator. This algorithm divides the variation process between two independent operators. The first operator, PG Variation, is used for directed performance improvement. The second operator, GA Variation, is used for divergent search.

In PGA-MAP-Elites, a single “greedy” policy (the actor) is trained alongside the critics to predict the action with the highest action-value for the critic update target, similar to the actor in the TD3 algorithm. However, unlike the actor in TD3, the greedy controller does not directly

interact with the environment. Instead experiences from the solution evaluations, which are stored as transitions in the replay buffer, are used for interaction with the environment. Asynchronously with each iteration of selection, variation, and evaluation, the critics are trained for a specified number of gradient descent steps and fitness gradient estimates are obtained immediately for any solution, without requiring any environment interaction by the solution for which the fitness gradient is being calculated. In particular, the fitness gradient estimate of each solution, ϕ is:

$$\nabla_{\phi} J(\phi) = \frac{1}{N} \sum \nabla_{\phi} \pi_{\phi}(s_t) \nabla_a Q_{\boldsymbol{\theta}}(s_t, a)|_{a=\pi_{\phi}(s_t)} \tag{2.32}$$

where s_t is the state sampled from the replay buffer, and $Q_{\boldsymbol{\theta}}$ is one of the greedily trained critics. However, to maintain the divergent search capabilities of the standard MAP-Elites, only half of the solutions sampled from the repertoire pass through the PG emitter, while the rest pass through the GA emitter (see Figure 2.7).

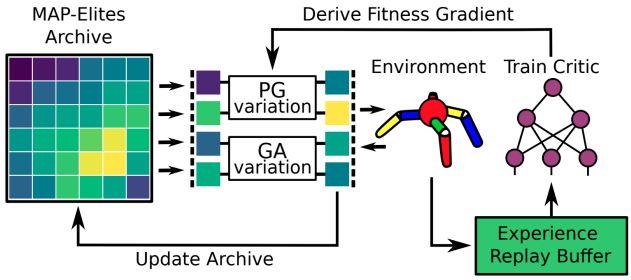


Figure 2.7: Main cycle of PGA-MAP-Elites. Adapted from [27].

To address the limitations of the PGA-MAP-Elites, particularly its shortcomings on tasks where the fitness objective directly discourages diversity in solutions, the Descriptor-Conditioned Gradients MAP-Elites (DCG-MAP-Elites) [29] algorithm extends the PGA-MAP-Elites by introducing two significant advancements. Firstly, it incorporates a descriptor-conditioned critic to direct PG optimisation to targeted descriptors, and secondly, it leverages the actor-critic training process to learn a descriptor-conditioned policy, effectively distilling the archive's knowledge into a single versatile policy capable of executing the full range of behaviours contained in the repertoire. A further update to this framework was made by DCG-MAP-Elites-AI [28], where the greedily trained descriptor-conditioned policy (versatile actor) is injected in the offspring during each generation, and consequently considered to be added to the repertoire. Interestingly, this update resulted to significant increase in the performance in all tasks evaluated compared to the algorithm's previous variant (see Figure 2.8).

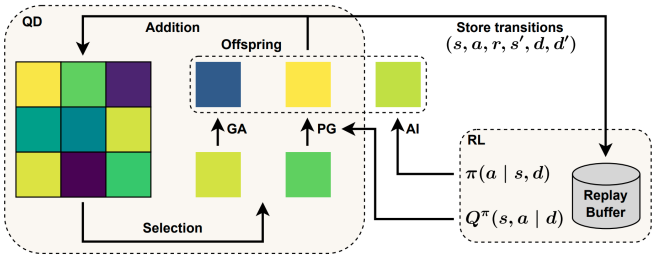


Figure 2.8: Main cycle of DCG-MAP-Elites-AI. Adapted from [28].

One can conceptualise PGA-MAP-Elites as a method that "greedily" trains a global actor and critic in a DRL conventional manner, that implicitly pulls the different solutions of the archive generated by GA, towards the global actor's behaviour, thereby improving their fitness. In contrast,

while DCG-MAP-Elites-AI also ”greedily” trains a global actor and critic, it conditions them on a specified descriptor. This approach aims to restrict the ”pull force” of the global actor and critic within the desired descriptor. This advancement not only improved the diversity of solutions in the repertoire but also significantly enhanced the fitness of previously low-performing solutions. It is now recognised as the state-of-the-art RL-QD algorithm, surpassing all competitors across all evaluated locomotion tasks.

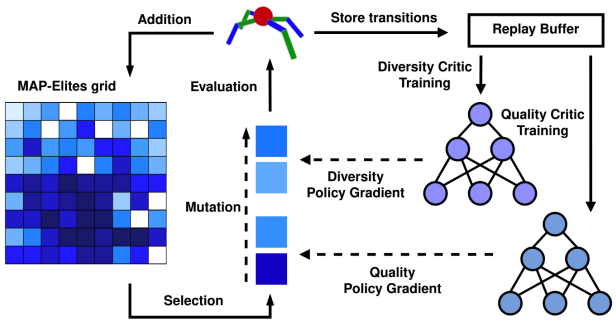


Figure 2.9: Main cycle of QD-PG. Adapted from [13].

Another algorithm is the QD-PG (Quality-Diversity Policy Gradient) [13] algorithm, which builds upon PGA-MAP Elites by replacing its GA variation operator with another PG operator, D-PG, to promote diversity. The main innovation in this algorithm is the creation of a time-step novelty reward function, which is used to compute time-step novelty rewards and calculate gradients using PG techniques. Additionally, to avoid local minima in exploration environments, the algorithm includes both a diversity-based and a quality-based critic. These critics are trained by all the solutions evaluated from the archive, in contrast to PGA-ME, which trains the critic using only one “greedy” actor (see Figure 2.9).

2.3.2 MAP-Elites with Evolution Strategies Based Emitter

MAP-Elites Evolution Strategies (ES) [35] algorithm replaces the variation operator with an ES method. In particular, every λ generations (MAP-Elites cycles), a solution is sampled sequentially from the MAP-Elites grid based on a mechanism that promotes novelty or quality, and is evolved for λ generation using an ES algorithm similar to Algorithm 3. In the highest-performing variant, EXPLORE-EXPLOIT ME-ES, the objective of the ES algorithm alternates between a novelty function and a fitness function. The novelty function $N(\boldsymbol{\theta}, k)$ calculates the average Euclidean distance between a solution’s behaviour descriptor and its k nearest neighbors in an archive, independent of the MAP-Elites grid (repertoire), that stores all previous solutions. Whereas the fitness function measures the episodic return when the agent follows the policy represented by the corresponding solution.

Building on the ME-ES, the MAP-Elites with Multiple Parallel Evolution Strategies (MEMES) [34] algorithm introduces several enhancements. The main change is that MEMES samples n number from the grid, and applies ES to each of one independently and in parallel. A proportion of these parallel optimisation processes (emitters) is dedicated to promoting diversity (explore), and the rest to promoting quality (exploit), similar to the ME-ES. Additionally, an innovative enhancement in MEMES is the use of a reset strategy, where each independent emitter can be reset separately based on its usefulness to the QD optimisation. This usefulness is measured by

tracking performance improvement with respect to the corresponding objective. By “reset”, we mean stopping the ES optimisation for a particular solution and resampling a new solution from the grid (see Figure 2.10).

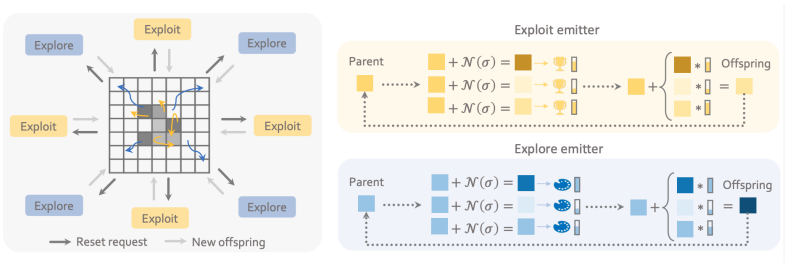


Figure 2.10: High-level pipeline of MEMES. Adapted from [34].

Chapter 3

Methodology

In this chapter, we detail the development of our final algorithm, discussing the challenges faced and decisions made. Specifically, we address the limitations encountered during the integration of the REINFORCE algorithm with MAP-Elites, modifications to the objective function, and the incorporation of a trajectory buffer. For each decision, we explain our rationale, drawing on relevant theories and previous research in the field.

3.1 Foundational Concepts and Operational Framework

In this section, we contextualise the theory discussed in the previous chapter and introduce some common vocabulary to facilitate clearer and easier explanations. Moreover, to avoid repetition, we discuss the components of the general RL & MAP-Elites (ME) pipeline that remain fixed throughout the development of the algorithm.

In this context, a '*genotype*' refers to a specific set of n parameters, $\boldsymbol{\theta} \in \mathbb{R}^n$, which define the configuration of a neural network, $\mathcal{N}$. This networks' architecture remains fixed, and the parameters $\boldsymbol{\theta}$ dictate the neural network's behaviour. When these parameters are implemented, the neural network generates a '*phenotype*' that embodies the agent's behaviour as it follows a policy $\pi_{\boldsymbol{\theta}}$. For clarity, we will collectively refer to the genotype, its correspodning phenotype, the policy, and the agent's resultant behaviour as a '*solution*'.

With these foundational principles established, we now turn our attention to the operational aspects of the algorithm. The selection mechanism begins by uniformly sampling $k \in \mathbb{N}$ solutions from the repertoire with replacement. We call k the batch size. These selected solutions are then divided equally, with half processed by the Genetic Algorithm (GA) emitter and the other half by the Reinforcement Learning (RL) emitter, which is the primary focus of this work and is thoroughly discussed in the next section. For the GA emitter, the selected solutions, referred to as 'parents,' undergo random mutations and/or crossover variations to produce their 'offspring,' which are then considered for addition to the repertoire. The addition mechanism follows the standard approach, where solutions are only added to the repertoire if the cell corresponding to their descriptor is empty or contains a less fit solution.

For a complete understanding, please refer to Figure 3.1 and Algorithm 4, which illustrate the main cycle of the RL-ME algorithm at a high level and present the pseudocode of the main algorithm, respectively. Additionally, Algorithms 5 and 6 provide the pseudocode for the Addition Mechanism and the Variation Operator, as depicted in Figure 3.1. Elements coloured in light blue are discussed thoroughly in the next section, while all other elements remain constant

throughout the development of our algorithm.

Notations and Their Meanings:

- $\mathcal{X}$ stores the solutions.
- $\mathcal{P}$ stores the fitness of the solutions.
- $\mathcal{S}$ stores some extra information of the solutions.
- p is the fitness value of a solution.
- $\boldsymbol{b}$ is the descriptor vector of a solution.
- $\boldsymbol{\theta}$ is the parent and $\boldsymbol{\theta}'$ its produced offspring.

Algorithm 4 RL-ME main algorithm.

procedure MAIN
  $(\mathcal{X}, \mathcal{P}, \mathcal{S}) \leftarrow$ create_empty_archive()
  $\mathcal{B} \leftarrow$ initialise_buffer()
  $i \leftarrow 0$
  while $i < I$ **do** ▷ Main Loop: I evaluations, batch-size k
    if $i < k$ **then** ▷ Orthogonal initialisation: k random $\boldsymbol{\theta}_i$
      $\pi_{\boldsymbol{\theta}'_{i+1}}, \ldots, \pi_{\boldsymbol{\theta}'_{i+k}} \leftarrow$ random_solutions()
    else ▷ Selection and variation
      $\pi_{\boldsymbol{\theta}'_{i+1}}, \ldots, \pi_{\boldsymbol{\theta}'_{i+k}} \leftarrow$ VARIATION$(k, \mathcal{X}, \mathcal{B})$
    end if
    ADD_TO_ARCHIVE$(\pi_{\boldsymbol{\theta}'_{i+1}}, \ldots, \pi_{\boldsymbol{\theta}'_{i+k}}, \mathcal{X}, \mathcal{P}, \mathcal{S})$
    $i \leftarrow i + k$
  end while
  return archive $(\mathcal{X}, \mathcal{P})$
end procedure

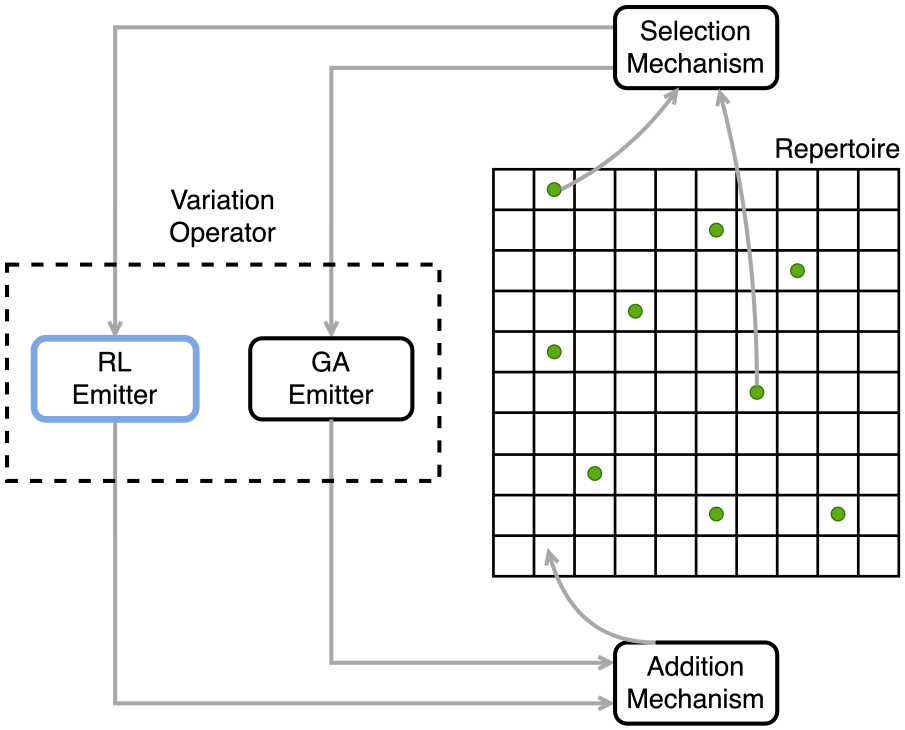


Figure 3.1: The RL & MAP-Elites pipeline

Algorithm 5 Addition Mechanism algorithm.

procedure ADD_TO_ARCHIVE(Solution-List, $\mathcal{X}, \mathcal{P}, \mathcal{B}$)
  for $\pi_{\boldsymbol{\theta}} \in$ Solution-List **do**
    $(p, \boldsymbol{b}, \text{trajectory}) \leftarrow \text{evaluate}(\pi_{\boldsymbol{\theta}})$
    processed_trajectory $\leftarrow$ PROCESS_TRAJECTORY(trajectory)
    add_to_buffer(processed_trajectory, $\mathcal{B}$)
    $c \leftarrow$ get_cell_index($\boldsymbol{b}$)
    if $(\mathcal{P}(c) = \text{empty}) \vee (\mathcal{P}(c) < p)$ **then**
      $\mathcal{P}(c) \leftarrow p, \mathcal{X}(c) \leftarrow \pi_{\boldsymbol{\theta}}$, $\mathcal{S}(c) \leftarrow \{\text{extra\_info}\}$
    end if
  end for
end procedure

Algorithm 6 Variation Operator algorithm.

procedure VARIATION($k, \mathcal{X}, \mathcal{B}$)
  for $i = 1$ **to** k **do**
    if $i \leq \lfloor \frac{k}{2} \rfloor$ **then**
      $\pi_{\boldsymbol{\theta}_a}, \pi_{\boldsymbol{\theta}_b} \leftarrow \text{selection}(\mathcal{X})$ ▷ Uniform sampling
      $\pi_{\boldsymbol{\theta}'_i} \leftarrow$ VARIATION_GA($\pi_{\boldsymbol{\theta}_a}, \pi_{\boldsymbol{\theta}_b}$)
    else
      $\pi_{\boldsymbol{\theta}_i}$, $\{\text{extra\_info}\} \leftarrow \text{selection}(\mathcal{X}, \mathcal{S})$ ▷ Uniform sampling
      $\pi_{\boldsymbol{\theta}'_i} \leftarrow$ VARIATION_PG($\pi_{\boldsymbol{\theta}_i}, \mathcal{B}, \{\text{extra\_info}\}$)
    end if
  end for
  return solutions $\pi_{\boldsymbol{\theta}'_1}, \ldots, \pi_{\boldsymbol{\theta}'_k}$
end procedure
procedure VARIATION_GA($\pi_{\boldsymbol{\theta}_1}, \pi_{\boldsymbol{\theta}_2}$)
  $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}_1 + \sigma_1 \mathcal{N}(\mathbf{0}, \boldsymbol{I}) + \sigma_2(\boldsymbol{\theta}_2 - \boldsymbol{\theta}_1)\mathcal{N}(\mathbf{0}, \boldsymbol{I})$
  return solution $\pi_{\boldsymbol{\theta}'}$
end procedure

3.2 Design

This section thoroughly discusses the development and design of the RL emitter depicted in Figure 3.1. More specifically, it covers all components highlighted in light blue in the pseudocode from the previous section, including VARIATION_PG$(\cdot)$, PROCESS_TRAJECTORY$(\cdot)$, and $\{$extra_info$\}$.

3.2.1 Integrating Monte Carlo Policy Gradient with MAP-Elites

The most straightforward integration of a Monte Carlo Policy Gradient (MCPG) method with MAP-Elites (ME) involves running the REINFORCE algorithm on each parent sampled from the repertoire and processed by the RL emitter. This was indeed our vanilla algorithm (see Figure 3.2 and Algorithm 7). In particular, during a generation, each parent in the RL emitter:

i. **Trajectory generation**: Generates a batch of $N \in \mathbb{N}$ trajectories, where each trajectory consists of a sequence of states, actions, and rewards obtained by following the current policy. Note that the trajectories are episodic.

ii. **Computation of standardised rewards-to-go**: For each trajectory, the rewards-to-go are computed at each time step and then standardised separately for each specific time step, maintaining the temporal distinctions within the data.

iii. **Data input to the objective function**: The actions, states, and standardised rewards-to-go from each transition in each trajectory are used to compute the REINFORCE objective function.

iv. **Policy update**: The gradient of the objective function with respect to the policy's parameters is computed. The parameters are then updated once (i.e., $n = 1$) in the direction of the gradient to enhance the policy's performance.

Algorithm 7 Vanilla MCPG Emitter.

procedure VARIATION_PG($\pi_{\boldsymbol{\theta}}, \gamma$)
  Generate a set of N episodes $\{S_0^i, A_0^i, R_1^i, \ldots, S_{T-1}^i, A_{T-1}^i, R_T^i\}_{i=1}^N$ following $\pi(\cdot|\cdot, \boldsymbol{\theta})$ ▷ i.
  $\{G_0'^i, \ldots, G_{T-1}'^i\}_{i=1}^N \leftarrow$ Standardise_Returns($\{R_1^i, \ldots, R_T^i\}_{i=1}^N, \gamma$) ▷ ii.
  Update $\boldsymbol{\theta}$ to $\boldsymbol{\theta}'$ using gradient ascent ▷ iii., iv.
  for $i = 1$ **to** n **do** ▷ n gradient steps
    $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=0}^{T-1} G_t'^i \nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}(a_t^i|s_t^i)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
  end for
  $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$
  return solution $\pi_{\boldsymbol{\theta}'}$
end procedure

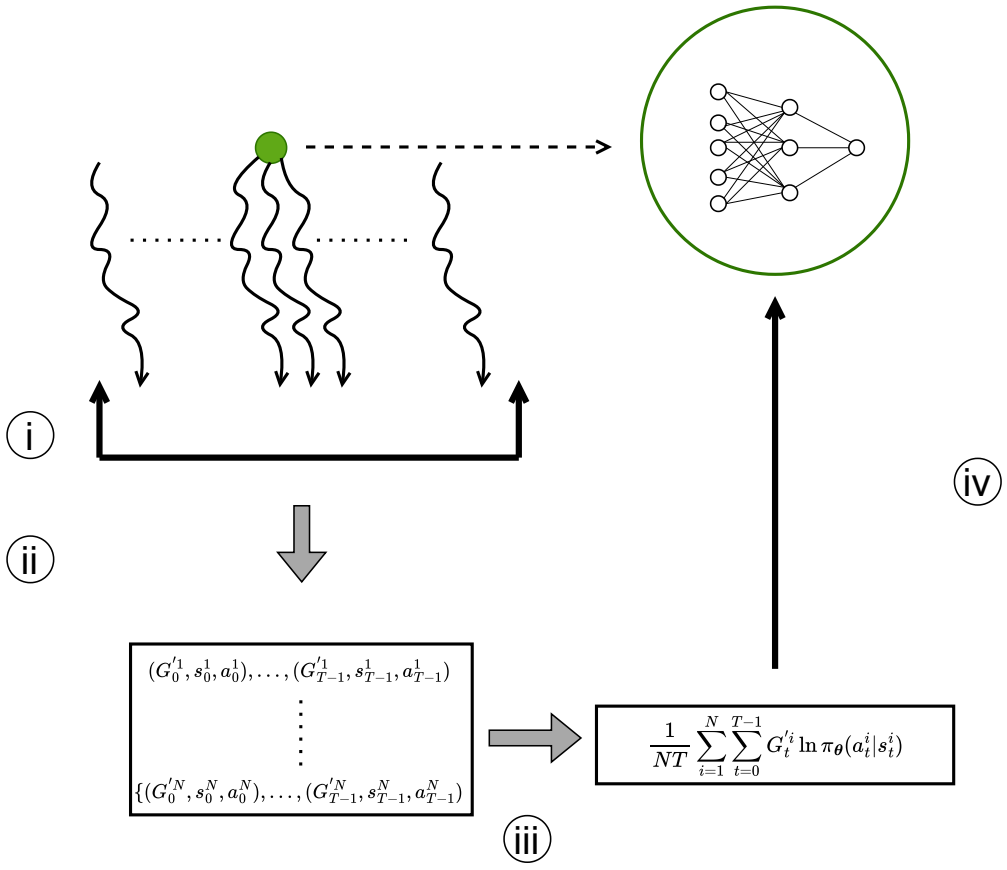


Figure 3.2: Vanilla MCPG emitter

The step ii. is primarily undertaken to reduce the variance of the estimation for $v(s_0)$—the REINFORCE objective—and thereby facilitate more stable learning. Note that by standardising the rewards-to-go along the time axis, as outlined in the Algorithm 8, involves subtracting a time-dependent baseline from them. To illustrate this, let G_t^i be the reward-to-go of time step t of the i^{th} episode. Then, the baseline subtracted from each reward-to-go is calculated as follows:

$$b_t = \frac{G_t^i(s_{G_t} - 1) + \bar{G}_t}{s_{G_t}} \tag{3.1}$$

where $\bar{G}_t = \frac{1}{N}\sum_{i=1}^{N} G_t^i$, and $s_{G_t} = \frac{1}{N-1}\sum_{i=1}^{N}(G_t^i - \bar{G}_t)^2$ are the sample mean and sample standard deviation of the set containing the rewards-to-go at time step t of each episode, $\{G_t^i\}_{i=1}^N$.

To clearer observe that, we have:

$$G_t^i - b_t = G_t^i - \frac{G_t^i(s_{G_t} - 1) + \bar{G}_t}{s_{G_t}} \tag{3.2}$$

$$G_t^i - b_t = \frac{s_{G_t}G_t^i - s_{G_t}G_t^i + G_t^i - \bar{G}_t}{s_{G_t}} \tag{3.3}$$

$$G_t^i - b_t = \frac{G_t^i - \bar{G}_t}{s_{G_t}} \tag{3.4}$$

As discussed in the background (see Section 2.1.7), this baseline is independent of the action taken at the corresponding time-step, a_t, ensuring the estimator remains unbiased. This method also reduces variance because it normalises the rewards' scale, which mitigates the impact of outliers or extreme values that can skew the learning process.

Furthermore, performing the standardisation along the time axis rather than across the entire dataset enables fairer comparisons of rewards-to-go at different times within an episode. This approach highlights which actions are performing above or below the average at specific moments, effectively converting the rewards-to-go into positive or negative adjustments to either encourage or discourage certain actions. Ignoring the temporal aspect would sometimes lead to misleading comparisons, such as equating the final reward-to-go, which represents only the terminal reward, with the reward-to-go at the beginning of the episode.

Algorithm 8 Computation of standardised rewards-to-go.

procedure STANDARDISE_RETURNS($\{R_1^i, \ldots, R_T^i\}_{i=1}^N, \gamma$)
  $\{G_0^i, \ldots, G_{T-1}^i\}_{i=1}^N \leftarrow$ COMPUTE_RETURNS($\{R_1^i, \ldots, R_T^i\}_{i=1}^N, \gamma$)
  for $t = 0$ **to** $T - 1$ **do**
    $s^2_{G_t} \leftarrow \mathrm{Var}(\{G_t^i\}_{i=1}^N)$
    $\bar{G}_t \leftarrow \mathrm{Mean}(\{G_t^i\}_{i=1}^N)$
    for $i = 1$ **to** N **do**
      $G_t'^i \leftarrow \frac{G_t^i - \bar{G}_t}{s_{G_t}}$
    end for
  end for
  return $\{G_0'^i, \ldots, G_{T-1}'^i\}_{i=1}^N$
end procedure
procedure COMPUTE_RETURNS($\{R_1^i, \ldots, R_T^i\}_{i=1}^N, \gamma$)
  for $i = 1$ **to** N **do**
    $G_T^i = 0$
    for $t = T - 1$ **to** 0 **do**
      $G_t^i \leftarrow R_{t+1}^i + \gamma G_{t+1}^i$
    end for
  end for
  return $\{G_0^i, \ldots, G_{T-1}^i\}_{i=1}^N$

3.2.2 Enhancing the Objective Function to Address Sample Inefficiency

Although REINFORCE can be highly time-efficient due to its ability to parallelise the processing of thousands of trajectories, it requires a substantial amount of data to achieve effective and stable learning. This substantial data requirement results in sample inefficiency, contradicting our goal of maintaining sample efficiency. REINFORCE's inefficiency stems from its Monte Carlo (MC) nature, which estimates value-state functions by averaging empirical returns from sampled trajectories. As discussed in Section 2.1.6 in the background, the variance of these estimators is inversely proportional to the number of samples; thus, a higher number of samples reduces variance and improves learning stability.

One enhancement for improving the sample efficiency involves the possibility of performing multiple update steps on the same batch of trajectories. However, this approach encounters significant challenges when applied to inherently on-policy algorithms such as REINFORCE. The fundamental issue is that each parameter update effectively changes the policy under which the data was generated. Specifically, the log probabilities essential to the objective function are recalculated for every gradient step, aligning them with the continuously updated policy. Therefore, this creates a scenario where updates are made using data from what becomes different policy—shifting the algorithm towards off-policy characteristics inadvertently.

It has also been stated by Schulman et al. [53] that performing multiple optimisation steps on an objective function similar to REINFORCE's is not well-defined and often results in excessively large policy updates that can destabilise learning.

Given the challenges identified with the traditional REINFORCE algorithm in managing sample inefficiency and policy stability with multiple gradient steps, we propose an enhanced approach modelled after the Proximal Policy Optimization (PPO) [53] algorithm. In particular, to counteract these issues and enhance stability, we integrate two key components to our objective

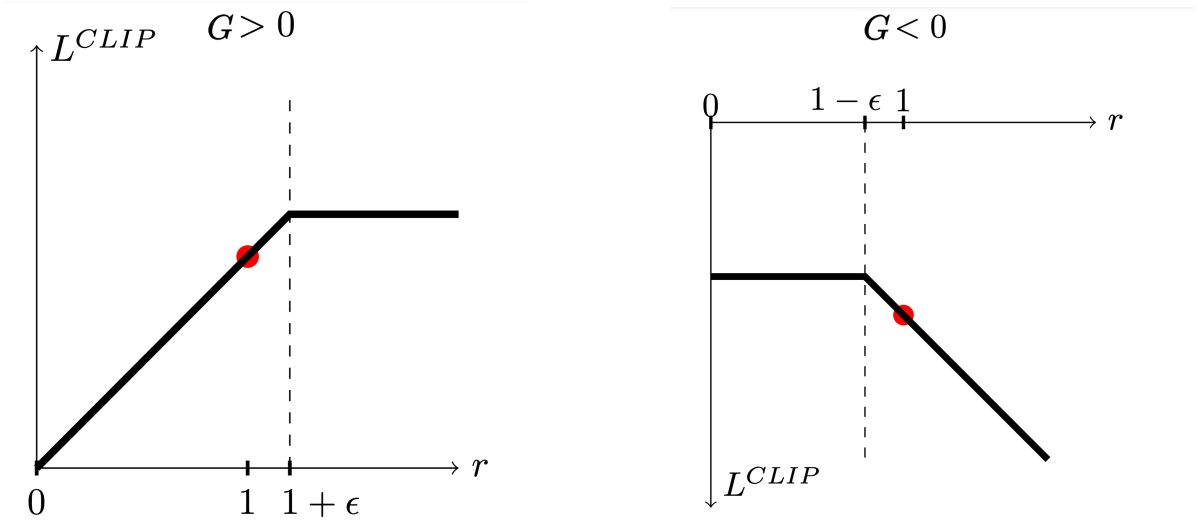


Figure 3.3: Plots showing a singe time-step of the surrogate function L^{CLIP} as a function of the sampling ratio r, for positive returns (left) and negative returns (right). The red circle on each plot shows the starting point for the optimisation, i.e. $r = 1$. Adapted from [53].

function and form a clipped surrogate objective similar to PPO's approach.

$$L^{\text{CLIP}}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\boldsymbol{\theta}) G'_t, \text{clip} \left(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon \right) G'_t \right) \right] \tag{3.5}$$

The first component denoted as $r_t(\cdot)$, employs importance sampling ratios, crucial for off-policy corrections. This technique adjusts the policy updates based on data generated by a slightly different policy, as discussed in section 2.1.6 of the background,

$$r_t(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(a_t|s_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(a_t|s_t)} \tag{3.6}$$

where $r_t(\boldsymbol{\theta}_{\text{old}}) = 1$.

The second component involves a clipping mechanism:

$$\text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon) \tag{3.7}$$

This mechanism caps the sampling ratio within the interval $[1-\epsilon, 1+\epsilon]$, removing incentives for excessive deviation from the old policy. We utilise $\epsilon = 0.2$, consistent with standard PPO implementations.

The final objective is determined by taking the minimum between the clipped and unclipped objectives, so that it is a lower bound on the unclipped objective. With this scheme, the change in probability ratio that would make the objective improve is ignored, while it is included when it makes the objective worse (see Figure 3.3).

Note that our objective differs from that of PPO only in that we use standardised time-step empirical returns, instead of the time-step advantage function estimator that PPO uses.

Therefore, with the upgraded objective function, and the increase in the number of gradient

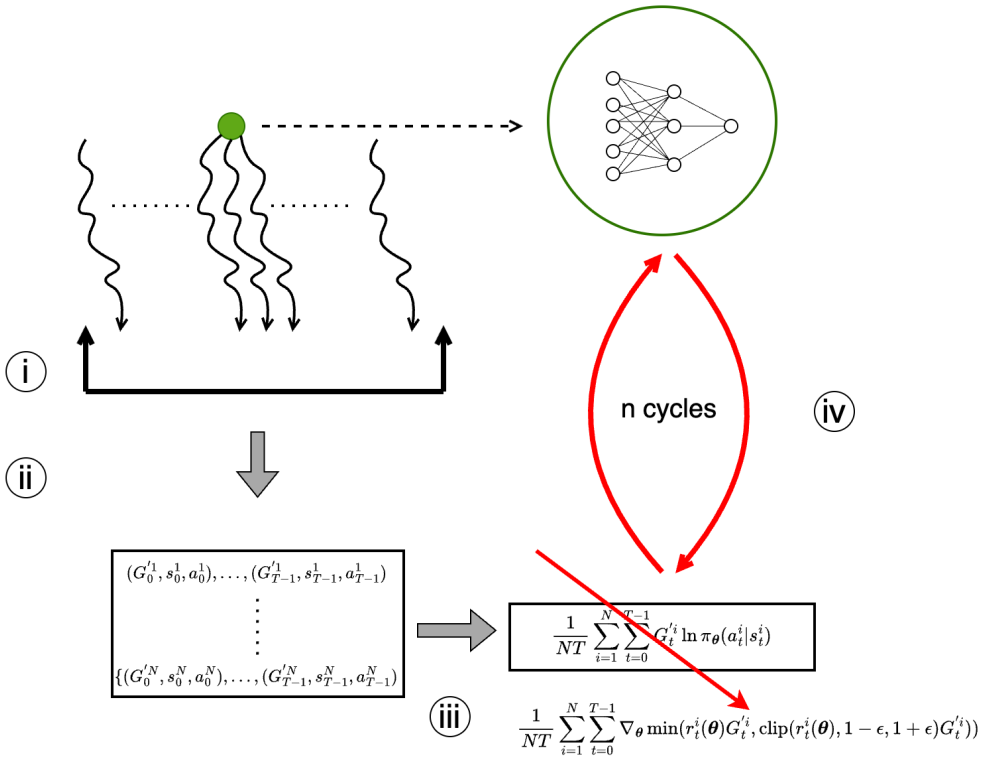


Figure 3.4: Vanilla MCPG emitter (multiple gradient steps).

steps, the procedure undertaken by the RL emitter (see Figure 3.4) includes two adjustments to the Algorithm 7:

1. **Multiple gradient steps**: $n > 1$ in line 5.

2. **Gradient of the objective function**: the gradient of the objective function is:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \min(r_t^i(\boldsymbol{\theta}) G_t'^i, \text{clip}(r_t^i(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon) G_t'^i) \tag{3.8}$$

3.2.3 Sample and Runtime Efficiency Challenges in the QD-RL Integration

While the increase in gradient steps and the modification of the objective function have demonstrated improved sample efficiency within a pure RL framework, these advantages did not fully translate to the QD-RL framework. Initial assessments of sample and runtime efficiency, compared to other baseline algorithms, showed our approach lagging in both areas. This inefficiency largely arises from significant overhead in each generation due to the generation of multiple trajectories.

This section outlines how data is utilised in our vanilla algorithm and the runtime of some of its processes. Understanding these aspects was crucial for the modifications made to develop the final algorithm discussed in the next section. For clarification, 'sample efficiency' refers to the algorithm's performance given a fixed number of evaluations. An 'evaluation' denotes the generation of an episodic trajectory by a solution, irrespective of its purpose.

3.2.3.1 Understanding the Emitters

The primary aim of QD is to uncover a variety of high-performing solutions. Diversity in our model mainly comes from the random mutations applied by the Genetic Algorithm (GA) emitter to the solutions in the repertoire. The RL emitter, on the other hand, is introduced to boost the fitness of these solutions, serving as a more a more powerful search method aimed at more efficiently optimising solutions. Here is how each emitter utilises evaluations:

1. **GA emitter**: Utilises $\mathbf{0.5} \times \mathbf{batch\_size} \times \mathbf{1}$ evaluations. $\mathbf{1}$ evaluation of each offspring for potential addition to the repertoire.

2. **RL emitter**: Utilises $\mathbf{0.5} \times \mathbf{batch\_size} \times (\boldsymbol{N}+\mathbf{1})$ evaluations. $\boldsymbol{N}$ evaluations for optimising each parent, and $\mathbf{1}$ evaluation of the corresponding offspring for potential addition to the repertoire.

With a fixed number of evaluations available, a clear trade-off emerges between optimising for high fitness via the RL emitter and enhancing diversity through the GA. A higher N could lead to more optimised solutions via the RL emitter but risks focusing too narrowly on a few high-fitness solutions at specific descriptors, neglecting the broader coverage strength of GAs.

3.2.3.2 Runtime Considerations

Further comparisons on the runtime of the variation processes in both emitters show that the runtime of non-evaluation processes, is minimal compared to the process needed to evaluate solutions (run episodic trajectories). To illustrate the time requirements, Figure 3.5 presents the time needed to generate 'Sample Size' trajectories (evaluate 'Sample Size' solutions) and to update the parameters of a genotype in the RL emitter, which is the most time-consuming process outside the process of evaluating the solutions.

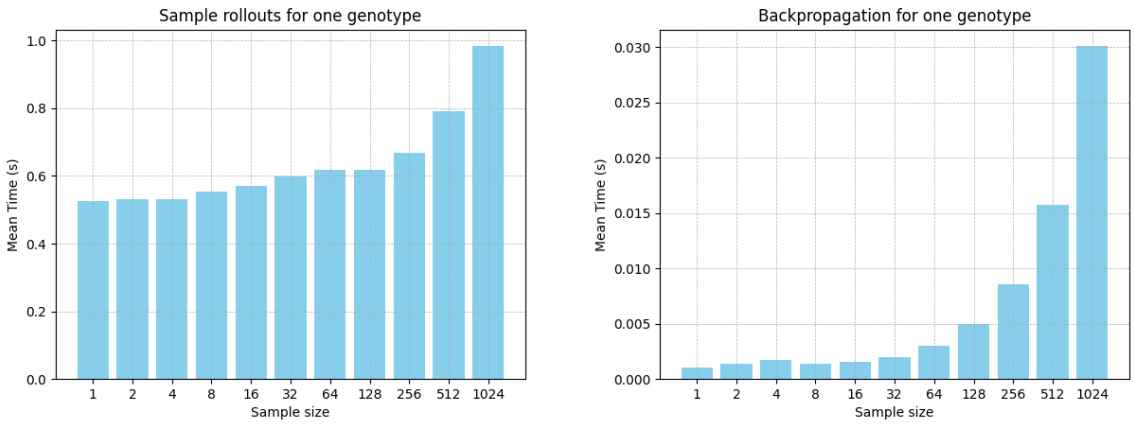


Figure 3.5: Plots showing the average time required over 10 repetitions to generate 'Sample Size' trajectories (left), and to update the parameters of a genotype processed through the RL emitter using data from 'Sample Size' trajectories (right). All experiments were conducted on the same GPU.

3.2.4 Final Algorithm: MCPG-ME

The insights from the previous section highlight the significant impact that evaluating solutions has on both the sample and runtime efficiency of our QD-RL algorithm. Consequently, a drastic change was necessary to utilise data more cleverly and efficiently.

3.2.4.1 Streamlining Data Utilisation

There are two reasons we need to evaluate the solutions:

1. **Optimisation**: Generate data to compute the objective function and produce the offspring.

2. **Addition to the repertoire**: Assess the quality and novelty of the solution for potential inclusion in the repertoire, which is essential for building and enhancing the collection of solutions in the repertoire.

Critical question: Can the Monte Carlo (MC) based framework be adjusted so that the same batch of data is used for both optimising the solutions and evaluating them for addition to the repertoire within the RL emitter?

Our response to this challenge is encapsulated in the development of the Monte Carlo Policy Gradient Map-Elites (MCPG-ME) algorithm, detailed in Algorithm 9 and illustrated in Figure 3.6. This also represents our final algorithm.

3.2.4.2 Operational Details of MCPG-ME

In each generation cycle, every solution processed by the MCPG emitter—the RL emitter of MCPG-ME—undergoes the following steps:

i. **Trajectory Sampling**: Randomly samples N trajectories from the offspring evaluations of the previous generation, stored in a buffer until the next generation's evaluation process.

ii. **Objective Function Computation**: Utilises the actions, states, and adjusted rewards-to-go from each transition in each trajectory to compute the objective function.

iii. **Policy Update**: Computes the gradient of the objective function and updates the solution parameters multiple times in the gradient's direction to produce the offspring.

iv. **Evaluation and Trajectory Buffer Update**: Each offspring is evaluated for addition to the repertoire, and its trajectory is added to the trajectory buffer. This process occurs in parallel for all offspring, and any previously stored trajectories in the buffer are simultaneously removed.

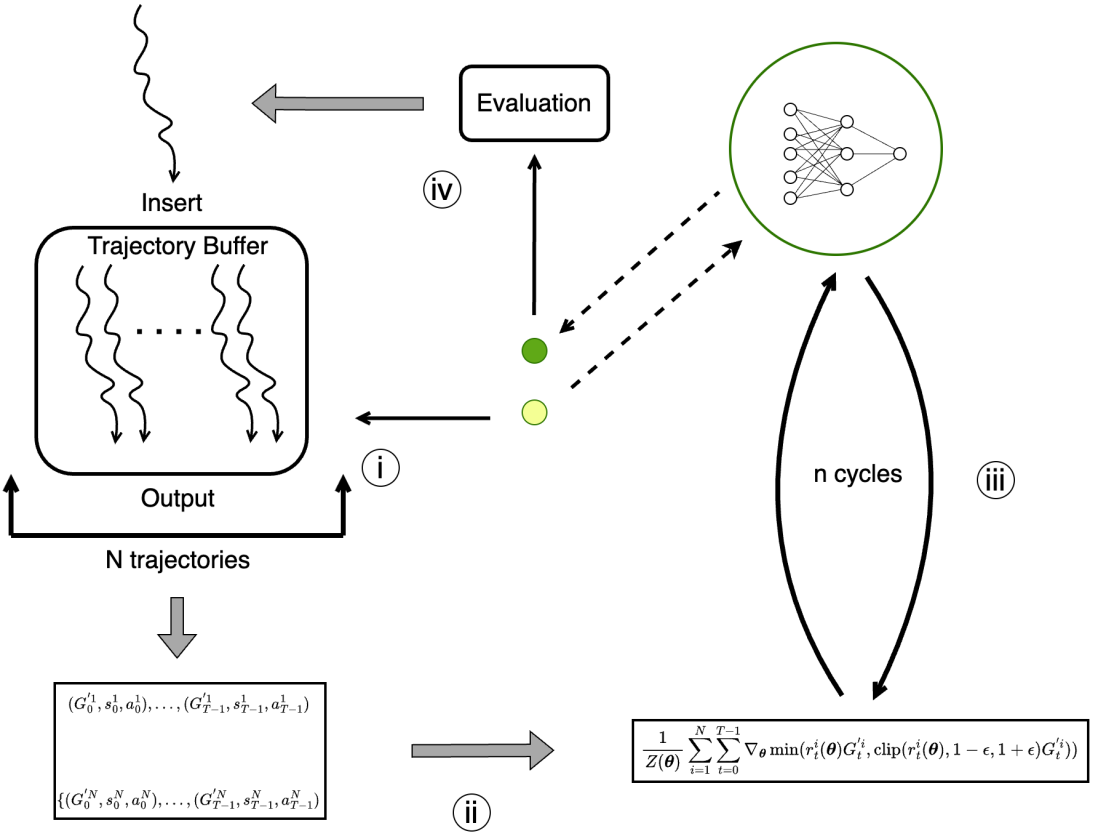


Figure 3.6: Final MCPG emitter.

During the development of this final version, three critical decisions regarding the processing and handling of data had to be made:

1. What baseline should be subtracted from the rewards-to-go in the objective function?
2. Do we sample transitions or trajectories from the buffer?
3. Which is the most effective method for sampling from the buffer?

1. **Baseline Selection:** Previously, we standardised the rewards-to-go along the time axis. However, in this framework, due to the variability in data coming from different policies—often not aligned with the target policy—this method proved impractical. Consequently, we now subtract the reward-to-go of the parent at each time-step, which is stored with the parent solution in the repertoire. This change serves as a reference to the target policy, highlighting the effectiveness of actions at specific time steps. Accordingly, the set labeled {extra_info} in Algorithms 5 and 6 contains the rewards-to-go associated with the parent. The process for handling these trajectories is detailed in Algorithm 10. Ideally, a state-dependent baseline would be subtracted as a variance reduction technique, providing more precise guidance. However, without a critic and given the continuous nature of the state space, implementing this approach presents significant challenges.

2. **Sampling Approach:** We have chosen to sample entire trajectories instead of individual transitions to enhance computational efficiency and simplify the process. This approach facilitates straightforward baseline subtraction by processing two arrays and avoids the complexities associated with identifying specific time steps. It also eliminates the need to record the time steps associated with each reward-to-go in the buffer.

3. **Sampling Strategy:** Despite our attempts to devise a more strategic approach, we found that random sampling remains the most effective method within the QD-RL framework. Avoiding randomness early in the process reduces the diversity of solutions in the repertoire, consequently impairing the final performance.

The algorithm is now entirely off-policy, as the target policy (the parent of the current generation), uses data generated by the offspring of the previous cycle's parents. As such, a further enhancement to the objective function has been incorporated, motivated from DD-OPG [54]—a trajectory-based off-policy algorithm. Specifically, we have replaced the normaliser constant of the objective function, previously set as $Z = N \times T$ with a new formulation: $Z(\boldsymbol{\theta}) = \sum_{i=1}^{N} \sum_{t=0}^{T-1} r_t^i(\boldsymbol{\theta})$, which is the sum over all the sampling ratios incorporated in the objective function. This approach, previously employed by Peshkin et al. [55], is both theoretically and empirically better behaved [56] [57] [58] than the standard normalising constant.

Algorithm 9 Final MCPG emitter algorithm.

procedure VARIATION_PG($\pi_{\boldsymbol{\theta}}, \mathcal{B}, \{G_t^p\}_{t=0}^{T-1}$)
  Sample N episodes, $\{S_0^i, A_0^i, R_1^i, \ldots, S_{T-1}^i, A_{T-1}^i, R_T^i\}_{i=1}^N$, from $\mathcal{B}$
  $\{G_0^i, \ldots, G_{T-1}^i\}_{i=1}^N \leftarrow$ COMPUTE_RETURNS($\{R_1^i, \ldots, R_T^i\}_{i=1}^N, \gamma$) ▷ See Algorithm 8
  Update $\boldsymbol{\theta}$ to $\boldsymbol{\theta}'$ using gradient descent
  for $i = 1$ **to** n **do** ▷ n gradient steps
    $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \leftarrow \nabla_{\boldsymbol{\theta}} \frac{1}{Z(\boldsymbol{\theta})} \sum_{i=1}^{N} \sum_{t=0}^{T-1} \min(r_t^i(\boldsymbol{\theta}) G_t'^i, \text{clip}(r_t^i(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon) G_t'^i$ ▷ where $G_t'^i = G_t^i - G_t^p$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \nabla J(\boldsymbol{\theta})$
  end for
  $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$
  return solution $\pi_{\boldsymbol{\theta}'}$

Algorithm 10 Process trajectory algorithm.

procedure PROCESS_TRAJECTORY(trajectory)
  rewards_to_go = COMPUTE_RETURNS(trajectory['rewards'])
  trajectory['rewards_to_go'] = rewards_to_go
  return trajectory

3.2.4.3 Source of Exploration

In our vanilla algorithm, the policies in the RL emitter transitioned between two distinct modes based on the phase of operation. During the evaluation of solutions for optimisation, the policy was stochastic, defined as

$$p(a_t|s_t;\boldsymbol{\theta}) = \mathcal{N}(a_t|\mu_{\boldsymbol{\theta}}(s_t), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}). \tag{3.9}$$

This stochastic nature was the source of exploration, and is a usual feature of on-policy learning. In contrast, during the evaluation of solutions for addition to the repertoire, the policy became deterministic, characterised by

$$p(a_t|s_t;\boldsymbol{\theta}) = \delta(a = \mu_{\boldsymbol{\theta}}(s_t)), \tag{3.10}$$

where δ represents the Dirac delta function.

The deterministic nature of the evaluation phase enhances the consistency and comparability of solutions and contributes to the reproducibility of behaviours stored in the repertoire, making the final product more robust. For example, given a specific state, s_t, a deterministic policy ensures that the same action is always taken subsequently. This consistency increases the likelihood of reproducing the same behaviour following time t.

For these reasons, policies remain deterministic during evaluations. However, this approach means that the data used for learning are also generated by deterministic policies, eliminating explicit action-based exploration previously enabled by stochastic policies. Nonetheless, exploration still occurs from two sources:

1. **Parameter-based Exploration**: On average, half of the parent policies sample trajectories generated by policies that underwent modifications through the GA emitter in the previous generation, introducing random variations in their parameters.

2. **Action-based Exploration**: Implicit exploration occurs through the objective function that incorporates importance sampling ratios. Even though the policy acts deterministically during evaluation, (3.10), the probabilities in the ratios are calculated as if the policy were stochastic, (3.9).

Chapter 4

Experiments

In this section, we briefly discuss the tools used to implement the MCPG-ME algorithm along with its hyperparameters. We detail the experimental setup, describing the QD tasks on which our algorithm was evaluated and the primary QD metrics utilised. We then present our experimental results and compare them to established baselines, which include other state-of-the-art algorithms, to assess our algorithm's effectiveness in achieving our objectives. Finally, we conduct an ablation study to evaluate the components added to MCPG's objective function.

We intend to answer the following three main questions:

1. Is MCPG-ME comparable in sample efficiency to the state-of-the-art algorithms documented in the literature?
2. How long does it take for MCPG-ME to train in comparison to its competitors and is it runtime-efficient compared to its competitors?
3. Is MCPG-ME scalable? Can massive parallelisation enhance final performance or reduce training time without compromising final performance?

4.1 Implementation

4.1.1 Tools

For the implementation of the algorithm, we utilise JAX [59] due to its robust capabilities for parallelisation and speed enhancement. JAX's architecture allows for just-in-time compilation and automatic differentiation, significantly accelerating computation and facilitating efficient scalability across various hardware accelerators such as GPUs. Alongside JAX, we employ QDax [31], an open-source library designed for QD optimisation. QDax offers a streamlined and modular API that supports a wide range of optimisation tasks from black-box optimisation to continuous control. Its integration with JAX ensures that all computational processes are optimally executed, leveraging JAX's parallelisation capabilities.

4.1.2 Neural Network Architecture and Hyperparameters

In all tasks, the policies are implemented using fully connected neural networks, each with two hidden layers of 64 neurons. The hidden layers use tanh activation functions, while the output layer employs a linear activation function.

The hyperparemeters of the MCPG-ME alogirthm are illustrated in Table 4.1.

Table 4.1: MCPG-ME hyperparameters

Parameter	Value
Number of centroids	1024
Total batch size k	512
MCPG batch size k_{MCPG}	256
GA batch size k_{GA}	256
Policy networks	[64, 64, $\|\mathcal{A}\|$]
GA variation param. 1 σ_1	0.005
GA variation param. 2 σ_2	0.05
MCPG gradient steps n	32
Policy learning rate	3×10^{-3}
Trajectory buffer size	512
Discount factor γ	0.99
Sample size N	1
Clipping parameter ϵ	0.2

4.2 Experimental Setup

Each experiment is replicated 10 times using random seeds, conducted across one million evaluations to assess the algorithm's sample and runtime efficiency, and extended to five million evaluations to test its scalability. We use the physics-based Brax simulator [60], which is optimised to run on and leverage JAX for fast, parallel computation, to simulate the environments. Additionally, all experiments are conducted using the same GPU, namely 'Nvidia Quadro RTX 6000". For quantitative results, we report p-values using the Wilcoxon-Mann-Whitney U test, with adjustments made using the Holm-Bonferroni correction.

4.2.1 Tasks

We evaluate MCPG-ME on five continuous control locomotion QD tasks as implemented in Brax and derived from standard RL benchmarks, see Table 4.2. AntTrap Omni and Ant Omni are *omnidirectional* tasks, where the simulated robot explores a predefined two-dimensional space defined by descriptor limits, aiming to minimise energy consumption. The fitness here is determined by the accumulated reward for surviving each time-step, penalised by energy usage, with the behavioral descriptor (BD) being the robot's final xy-position. In contrast, Ant Uni, Walker Uni, and Hopper Uni are *unidirectional* tasks focused on discovering diverse locomotion strategies while balancing speed against energy consumption. Fitness for these tasks is defined by the forward progress made, combined with survival time and an energy consumption penalty, with the BD being the contact rate of each foot. Task states include measurements like the center of gravity's height, (x,y,z)-velocities, roll, pitch and yaw angles, and the relative position of the joints of the robot. Actions are delivered as continuous-valued torques applied to control the robot's joints, with initial joint positions sampled from a Gaussian distribution, adding a stochastic element to the tasks.

These tasks provide a holistic evaluation in the sense that they asses the algorithm's ability to discover a diverse set of high-perofrming behaviours where:

- **Unidirectional tasks**: The fitness objective does *not directly discourage* diversity in behaviours.
- **Ant Omni**: The fitness objective does *directly discourage* diversity in behaviours, but the

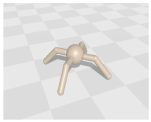
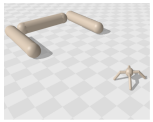
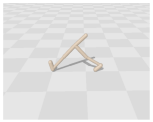
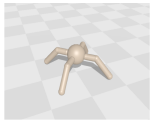
environment is *not deceptive*.

- **AntTrap Omni**: The fitness objective *directly discourages* diversity in behaviours, and the environment is *deceptive*.

The reason the fitness objective of the omnidirectional tasks discourages diversity is that the global maximum of the fitness function is achieved when the robot remains stationary, minimising energy consumption, which directly opposes the goal of exploring different xy-positions. Furthermore, the AntTrap Omni features a deceptive environment with a trap in front of the robot's initial position, preventing it from discovering new locations if it heads straight towards the trap. This trap also creates a discontinuity in the BD space, where points on either side of the trap are equally accessible but require drastically different trajectories [28].

In each unidirectional task, we evaluate our algorithm using episodes of lengths 1000 steps and 250 steps to observe any performance differences compared to other baselines. We adopt this approach because, as mentioned in the background chapter (see Section 2.1.6), the necessity for complete episodes means that Monte Carlo methods can be a limitation in environments where episodes are longer or where reaching a terminal state is rare, delaying policy improvement. For the omnidirectional tasks, we limit episodes to a length of 250 steps to maintain the challenge of the task. Allowing longer episodes would make it easier for the robot to explore more xy-positions. For simplicity, we denote each environment by its name followed by the episode length in brackets, e.g., Ant Uni (1000) for Ant Uni with 1000-step episodes.

Table 4.2: Evaluation Tasks

	Ant Omni	AntTrap Omni	Walker Uni	Hopper Uni	Ant Uni
					
State	Position and velocity of body parts				
Action	Torques applied at the hinge joints				
State dim	30	30	18	12	28
Action dim	8	8	6	3	8
Descriptor dim	2	2	2	1	4
Episode len	250	250	250 & 1000	250 & 1000	250 & 1000
Parameters	6672	6672	5772	5190	6544

4.2.2 Baselines

We compare MCPG-ME with four state-of-the-art algorithms: MAP-Elites [14], MEMES [34], PGA-MAP-Elites [27], and DCG-MAP-Elites-AI [28]. We have chosen MAP-Elites as one of our baselines because it forms the foundation of our algorithm and is recognised as the fastest to train among QD algorithms. Therefore, it provides a solid benchmark for assessing the training speed of our algorithm relative to the fastest available. Moreover, MEMES was chosen due to its similarities with MCPG-ME. First, MEMES employs Natural Evolution Strategies, which share strong similarities with Monte Carlo-based methods. Secondly, each solution sampled from the repertoire is optimised independently, without relying on any global heuristic information. PGA-MAP-Elites, an actor-critic-based MAP-Elites algorithm, stands out as the most

sample-efficient performer in all unidirectional locomotion tasks among QD algorithms that do not utilise information from the BD of solutions for optimisation purposes. Given that no work has been conducted on integrating BD information into our pipeline, PGA-MAP-Elites serves as an excellent benchmark for assessing sample efficiency to a state-of-the-art algorithm fairly. Finally, we compare against DCG-MAP-Elites-AI, an extension of PGA-MAP-Elites that incorporates descriptor-conditioned training and emerges as the highest-performing sample-efficient algorithm across all omnidirectional and unidirectional locomotion tasks (with MDP assumptions) among the QD algorithms in the literature. The hyperparameters of each baseline aglorithm can be found in Appendix .1.

4.2.3 Metrics

To evaluate the final archives (repertoires) of all algorithms throughout training, we consider the following three metrics, as defined by Flageat et al. [61], and used in the relevant papers in the literature:

- **QD Score**[5]: The sum of the fitness, f_i of all individuals in an archive $\mathcal{A}$. It evaluates both the quality and diversity of $\mathcal{A}$:

$$\mathrm{QDScore}(\mathcal{A}) = \sum_{i=1}^{N_{\mathcal{A}}} f_i \tag{4.1}$$

- **Coverage**: The proportion of filled cells in the archive. It estimates the diversity present in $\mathcal{A}$ by measuring the BD space illumination.

$$\mathrm{Coverage}(\mathcal{A}) = \frac{N_{\mathcal{A}}}{\text{Number of cells}} \tag{4.2}$$

- **Max Fitness**: The highest achieved fitness among all individuals present in an archive $\mathcal{A}$.

$$\mathrm{MaxFitness}(\mathcal{A}) = \max_{1 \leq i \leq N_{\mathcal{A}}}(f_i) \tag{4.3}$$

4.3 Main Results

4.3.1 Sample Efficiency

The experimental results presented in Figure 4.1 demonstrate that although the QD score of MCPG-ME is lower than that of DCG-MAP-Elites-AI in omnidirectional tasks, it is significantly higher than those of the other baselines ($p < 0.01$), including MAP-Elites. As shown in the coverage plots, this outcome is primarily due to MCPG-ME's ability to explore a wide range of diverse solutions across each task very quickly, covering nearly the entire repertoire with very few evaluations. Since the MCPG emitter is the only component added to MAP-Elites to create MCPG-ME, these results suggest that it significantly enhances diversity in deceptive environments such as AntTrap Omni, and in tasks where the fitness objective discourages diversity, like both omnidirectional tasks. The final repertoires for all algorithms in Ant Omni and AntTrap Omni can be seen in Figures 4.2 and 4.3 respectively.

Turning to the unidirectional tasks of 250-step episodes depicted in Figure 4.1, MCPG-ME continues to excel in finding diversity as achieves a significantly higher coverage than the other baselines, with $p < 0.01$. Additionally, it achieves significantly higher QD score than all the baselines in the Walker Uni (250) and Hopper Uni (250) tasks, with $p < 0.01$. This difference in

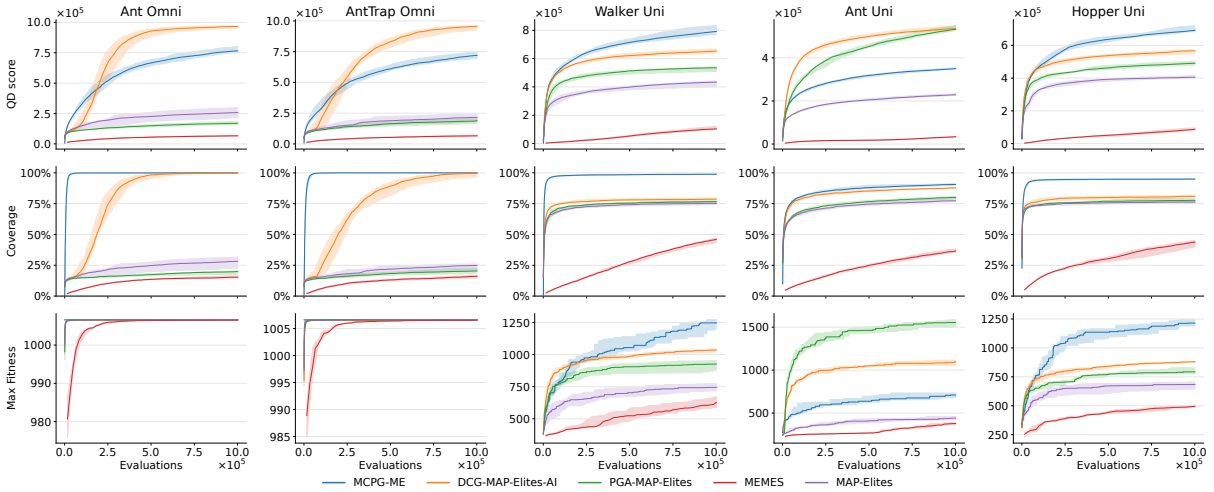


Figure 4.1: QD score, coverage and max fitness (Section 4.2.2) with respect to the number of evaluations for MCPG-ME and all baselines on all **250-step** tasks. Each experiment is replicated 10 times with random seeds. The solid line is the median and the shaded area represents the first and third quartiles.

performance is likely due to both higher performing solutions and a greater number of solutions in the repertoire, as evidenced by increased max fitness and coverage. Indeed, this pattern is confirmed in the Walker Uni (250) task, as shown in Figure 4.5. However, in the Ant Uni task, despite achieving a higher QD score than MAP-Elites and MEMES, MCPG-ME performs worse than PGA and DCG-MAP-Elites. Given that MCPG-ME's coverage continues to outperform all baselines with a significance level of $p < 0.01$, its inferior performance is likely attributed to difficulties in efficiently finding high-performing solutions. The Ant Uni (250) task is inherently more challenging than the other unidirectional tasks, as it features action and state spaces of higher dimensions (see Table 4.2). Therefore, the difference in performance might stem from the injection of high-perfoming solutions in the repertoire ('greedy' actor) and the use of global critics in actor-critic based methods, which perform value assessments and speed up learning while reducing the amount of data required to achieve significant learning progress.

Interestingly, the experimental results presented in Figure 4.4 show that in unidirectional tasks with 1000-step episodes, MCPG-ME continues to perform competitively in terms of coverage. It outperforms all baselines across all metrics in the Hopper Uni (1000) task ($p < 0.05$), except when compared to DCG-MAP-Elites-AI in max fitness, where the results are not significant. Yet, in the Walker Uni (1000) task, it achieves a lower QD score than DCG-MAP-Elites-AI and a score similar to that of PGA-MAP-Elites, mainly due to its higher coverage as seen in Figure 4.6. The difference in the relative performance of MCPG-ME compared to the actor-critic based methods between 250-step and 1000-step episodes might be attributed to the fact that the longer the episode in the Walker Uni (1000) task, the higher the chance of termination before episode completion. These early terminations lead to wasted data, as MCPG-ME learns from episodic trajectories, and any data collected after the episode terminates is ignored. This contrasts with the actor-critic based algorithms, which continue to utilise data from the newly started incomplete episode even after an episode is terminated. In general, the relative performance compared to the actor-critic based algorithms has slightly worsened in 1000-step episodes compared to 250-step episodes, suggesting a potential decrease in sample efficiency as episode length increases and thus aligning with the Monte Carlo theory mentioned in Section 2.1.6.

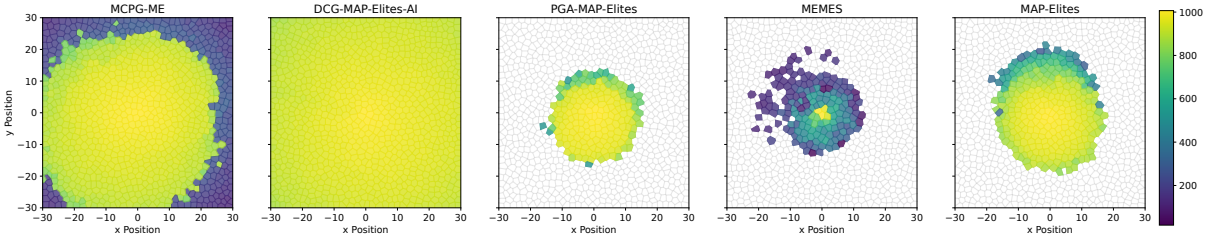


Figure 4.2: Ant Omni (250) Repertoire at the end of training for all algorithms.

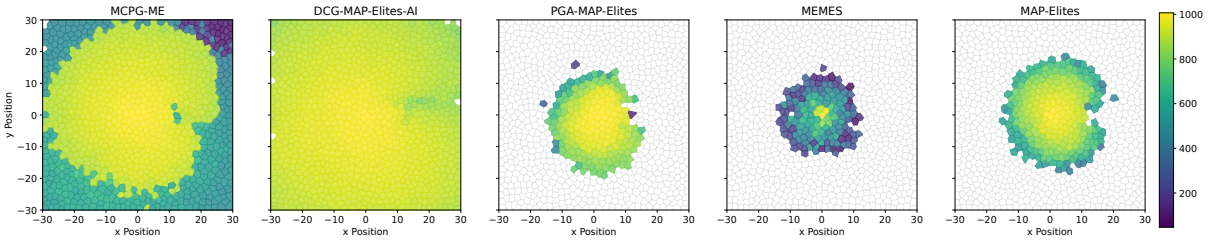


Figure 4.3: AntTrap Omni (250) Repertoire at the end of training for all algorithms.

Overall, MCPG-ME's early strong coverage likely serves as the primary boost to the QD score. This is because of the substantial difference between having even a bad solution in a cell of the repertoire and having an empty cell. The presence of a bad solution enhances the QD score in two significant ways: (1) it can improve the solution in the cell over generations, (2) it can produce offspring that may be placed in other cells, thereby perpetuating a cycle of continuous improvement where each new solution potentially initiates further optimisation across many generations.

4.3.1.1 Hypothesis for Achievements in Coverage

Achievements in coverage by MCPG-ME may stem from learning instabilities exhibited by some policies when processed in the MCPG emitter early in the process. These instabilities primarily arise from a large variance in the objective function, which is introduced by significant and constant deviations between the behaviour policy and the target policy, leading to substantial updates in the parameters of the target policy. Such large updates can be likened to random mutations in the solution parameters, which are larger than those typically seen in MAP-Elites, potentially explaining the significant differences in coverage between MCPG-ME and MAP-Elites.

4.3.1.2 Sample Efficiency Evaluation

The experimental results presented in Figures 4.1 and 4.4 demonstrate that MCPG-ME excels at efficiently finding diverse solutions, covering the final repertoire to a similar or greater extent than all the baselines across all evaluated locomotion tasks. This ability seems to be the key driver of its high QD score in many tasks, as discussed previously. Moreover, it achieves a significantly higher QD score ($p < 0.01$) than both MAP-Elites and MEMES in all tasks, indicating the MCPG emitter's success in more sample-efficiently finding a diverse set of high-performing solutions when compared to algorithms that do not use critics. However, it is important to note that MEMES, due to not using Markov Decision Process (MDP) assumptions, is expected to be much less sample-efficient, as time-step information of the evaluations (trajectories) is not utilised. Moreover, when compared to PGA-MAP-Elites, except for the Ant Uni tasks, MCPG-ME achieves a similar or higher QD score, indicating that critics are not necessarily required for a competitive, sample-efficient QD algorithm in relatively easy locomotion tasks. Finally, MCPG-ME performs

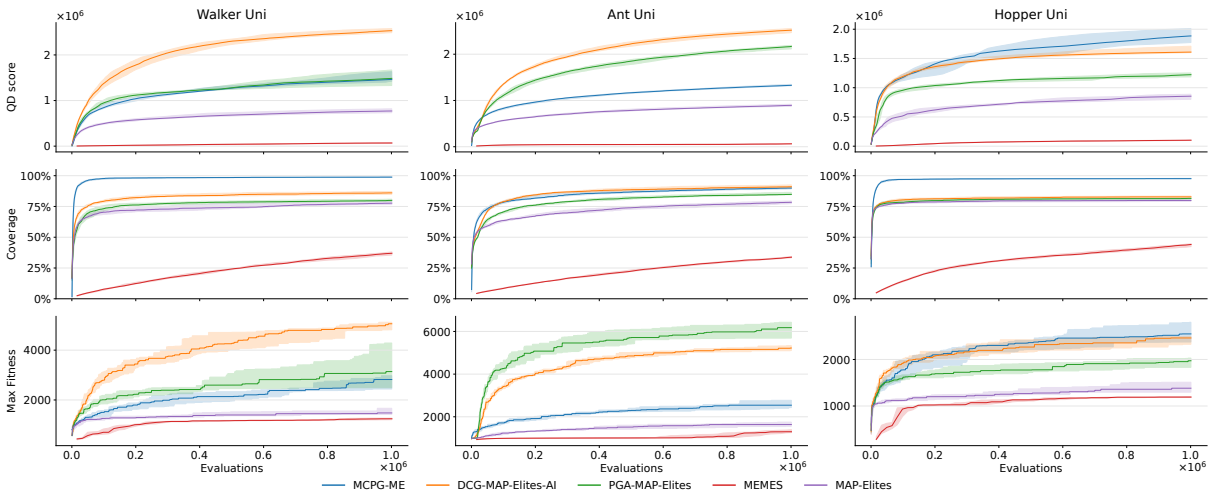


Figure 4.4: QD score, coverage and max fitness (Section 4.2.2) with respect to the number of evaluations for MCPG-ME and all baselines on all **1000-step** tasks. Each experiment is replicated 10 times with random seeds. The solid line is the median and the shaded area represents the first and third quartiles.

competitively compared to DCG-MAP-Elites-AI in some tasks, sometimes achieving a higher QD score depending on the episode length. On average, it is clear that DCG-MAP-Elites-AI is more sample-efficient than MCPG-ME in the omnidirectional tasks and the more challenging unidirectional tasks. Given that the main difference between DCG-MAP-Elites-AI and PGA-MAP-Elites is the descriptor-conditioned training, this advantage is primarily due to DCG-MAP-Elites-AI utilising descriptor information for solution optimisation, placing it in a more data-informed and advantageous position compared to MCPG-ME.

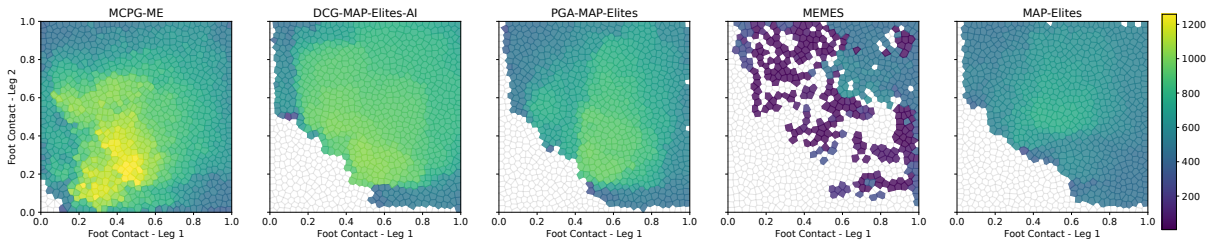


Figure 4.5: Walker Uni (250) Repertoire at the end of training for all algorithms.

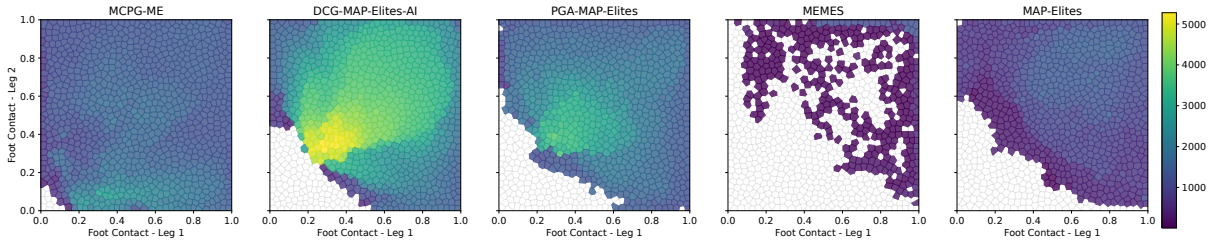


Figure 4.6: Walker Uni (1000) Repertoire at the end of training for all algorithms.

4.3.2 Runtime

As presented in the experimental results in Figures 4.7 and 4.8, excluding MAP-Elites, the runtime of MCPG-ME is significantly lower than all the baselines across all tasks and both episode length variants ($p < 0.01$). It is, in fact, roughly four times faster than the fastest actor-critic-based method (DCG-MAP-Elites-AI) in all 250-step episode tasks, and roughly twice as fast in all 1000-step episode tasks. Aligning with expectations, MAP-Elites is the fastest in all tasks; however, MCPG-ME's runtime is close, being no more than 1.5 times slower than MAP-Elites in any of the tasks evaluated.

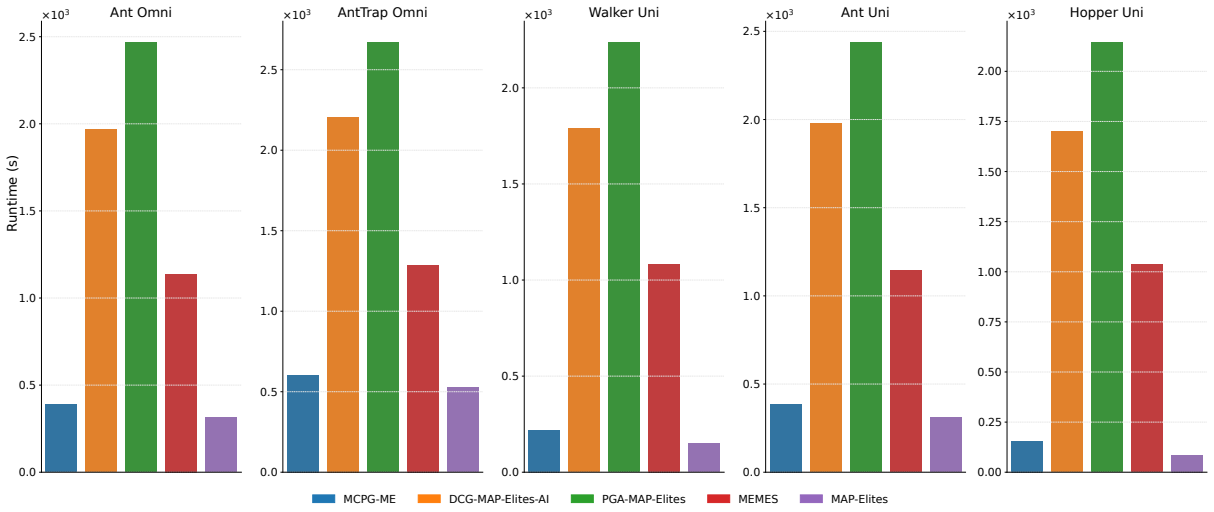


Figure 4.7: Runtime of each algorithm on the **250-episode** tasks.

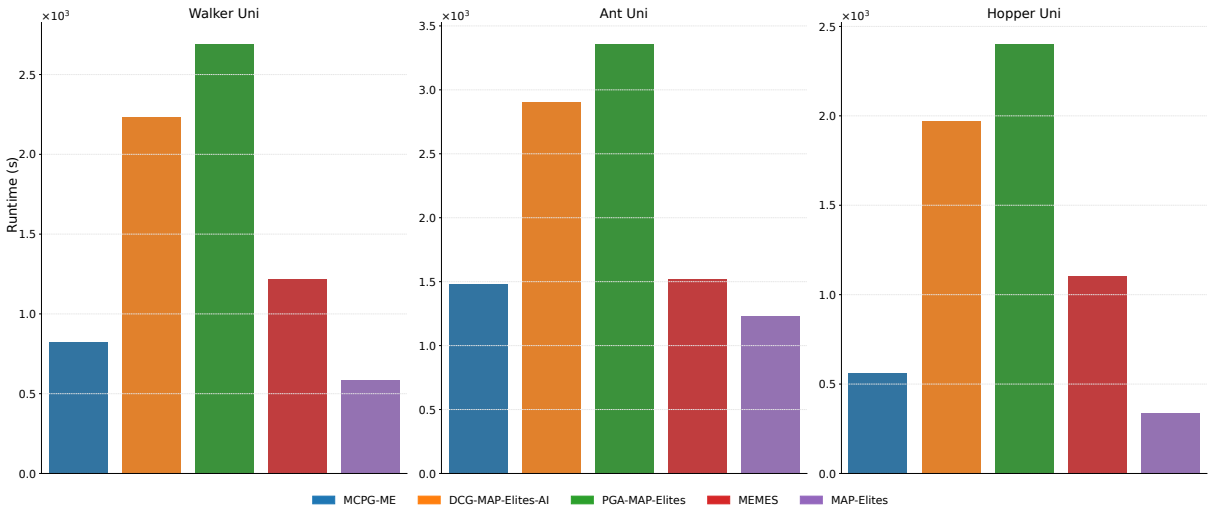


Figure 4.8: Runtime of each algorithm on the **1000-episode** tasks.

4.3.2.1 Runtime Efficiency

To evaluate the runtime efficiency of MCPG-ME, we consider its lowest final QD score from the 10 replications of each experiment conducted in the previous section on sample efficiency. Then, we compare the median times required by MCPG-ME and the other baseline algorithms to achieve this score. This evaluation approach could alternatively focus on whether MCPG-ME can achieve the baselines' QD scores in less time. However, such an analysis might not interest

those prioritising both sample and runtime efficiency. For instance, in some tasks, MCPG-ME might match the training times of its competitors but use over four times as much data. In contrast, our evaluation complements the previous section by determining whether MCPG-ME's QD score—achieved after one million evaluations—is competitive with state-of-the-art algorithms (outlined in the previous section), and whether it is attained more or less quickly.

We exclude MAP-Elites from these comparisons because, although it is faster, its highest final QD score is significantly lower ($p < 0.01$) than that of MCPG-ME across all tasks. Moreover, given the trend of its QD score as shown in Figures 4.1 and 4.4, it is very unlikely that MAP-Elites would achieve MCPG-ME's final score more quickly. Additionally, no comparisons are needed for MEMES, as its highest final QD score is lower in all tasks and it is slower than MCPG-ME, with all these results having a significance level of $p < 0.01$. The same applies to PGA-MAP-Elites in all tasks except for Ant Uni, as MCPG-ME achieves a higher or comparable final median QD score while being at least twice as fast. Following the same criteria, the only comparisons needed are with DCG-MAP-Elites-AI on the omnidirectional tasks and the Walker Uni (1000) task, as well as with both DCG-MAP-Elites-AI and PGA-MAP-Elites on the Ant Uni tasks.

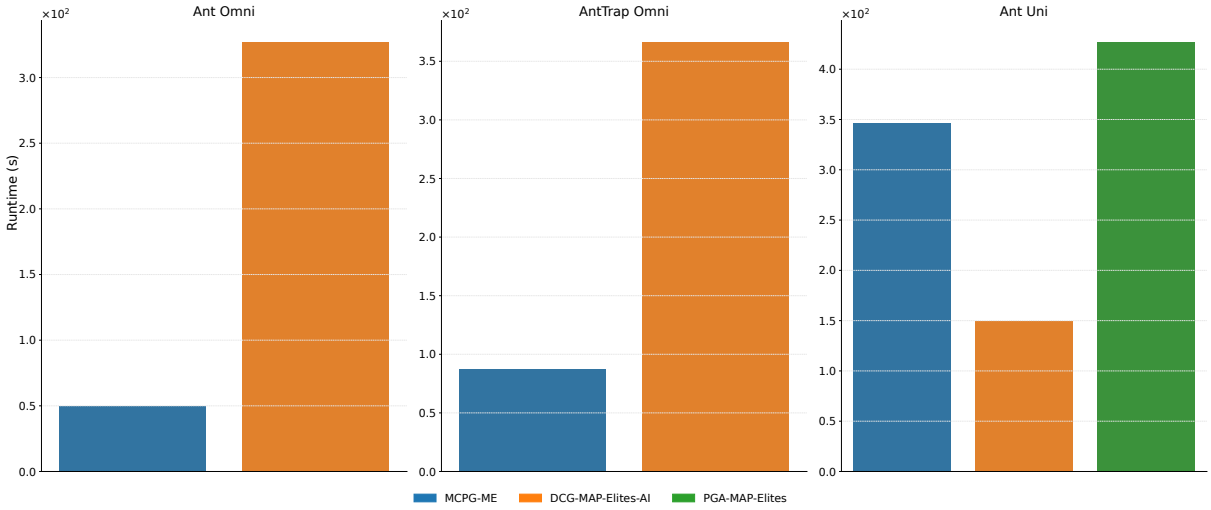


Figure 4.9: Time required for each algorithm to achieve MCPG-ME's lowest QD score in the **250-step** tasks.

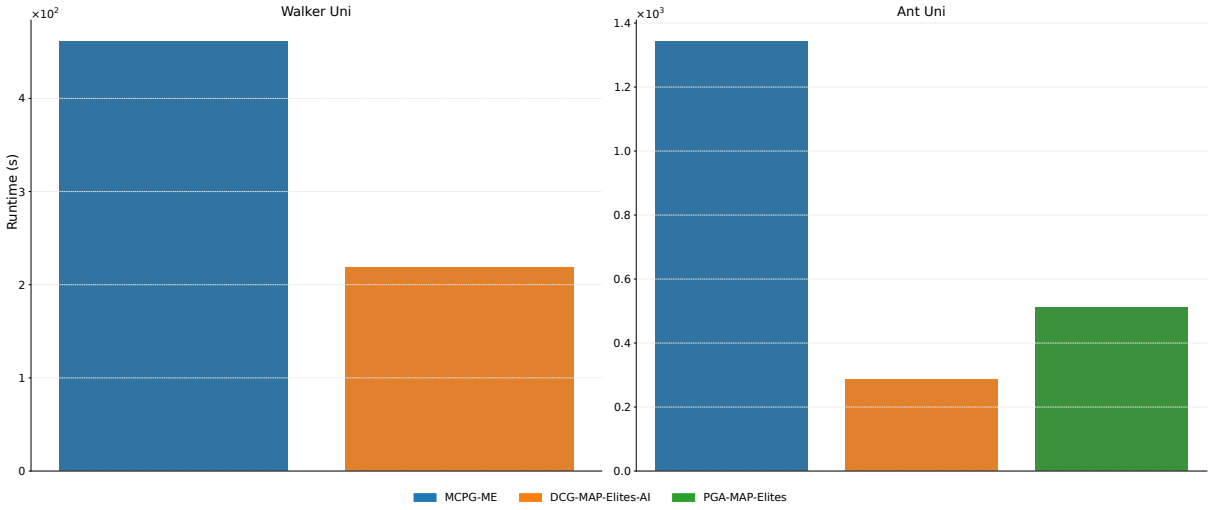


Figure 4.10: Time required for each algorithm to achieve MCPG-ME's lowest QD score in the **1000-step** tasks.

Having as a reference the lowest QD score achieved by MCPG-ME in each task, the analysis presented in Figures 4.9 and 4.10 demonstrates that MCPG-ME is significantly more time-efficient than DCG-MAP-Elites in both omnidirectional tasks. While MCPG-ME outperforms PGA-MAP-Elites in terms of time efficiency in the Ant Uni (250), it is less time-efficient than DCG-MAP-Elites in this setting. In Ant Uni (1000) MCPG-ME is less time-efficient than both actor-critic based algorithms and in Walker Uni (1000) it is less time-efficient than DCG-MAP-Elites-AI. All reported comparisons are statistically significant, with a significance level of $p < 0.01$.

4.3.2.2 Runtime Efficiency Evaluation

To conclude, MCPG-ME achieves its lowest QD score significantly faster than all baselines across most evaluated tasks, with exceptions including comparisons with PGA-MAP-Elites in the Ant Uni task (1000) and with DCG-MAP-Elites-AI in the Ant Uni tasks (250, 1000) and the Walker Uni task (1000).

4.3.3 Scalability

To evaluate the scalability of MCPG-ME, we test how massive parallelisation affects its performance. In our framework, a straightforward way to control parallelisation is by varying the number of batch size, $N_{\mathcal{B}}$, i.e., the number of solutions involved in each generation, as each solution in each emitter is processed in parallel if the hardware accelerator allows. For this reason, the experimental setup differs slightly from previous sections and instead follows a similar approach used by Lim et al. [33]. Specifically, we vary $N_{\mathcal{B}}$ and plot the QD score with respect to the number of evaluations and iterations (generations) as well as the runtime of the experiment for each $N_{\mathcal{B}}$ across all evaluated tasks.

4.3.3.1 Scalability of MCPG-ME

Each experiment is replicated 10 times using random seeds, over five million evaluations. Due to time constraints, MCPG-ME is evaluated only on Ant Uni (250), Walker Uni (250), and Ant Omni (250). Since Hopper Uni (250) and Anttrap Omni (250), share many similarities with Walker Uni (250) and Ant Omni (250) respectively, in terms of state and action space dimensions as well as the nature of the tasks, it was deemed reasonable to exclude them as significant differences were not expected.

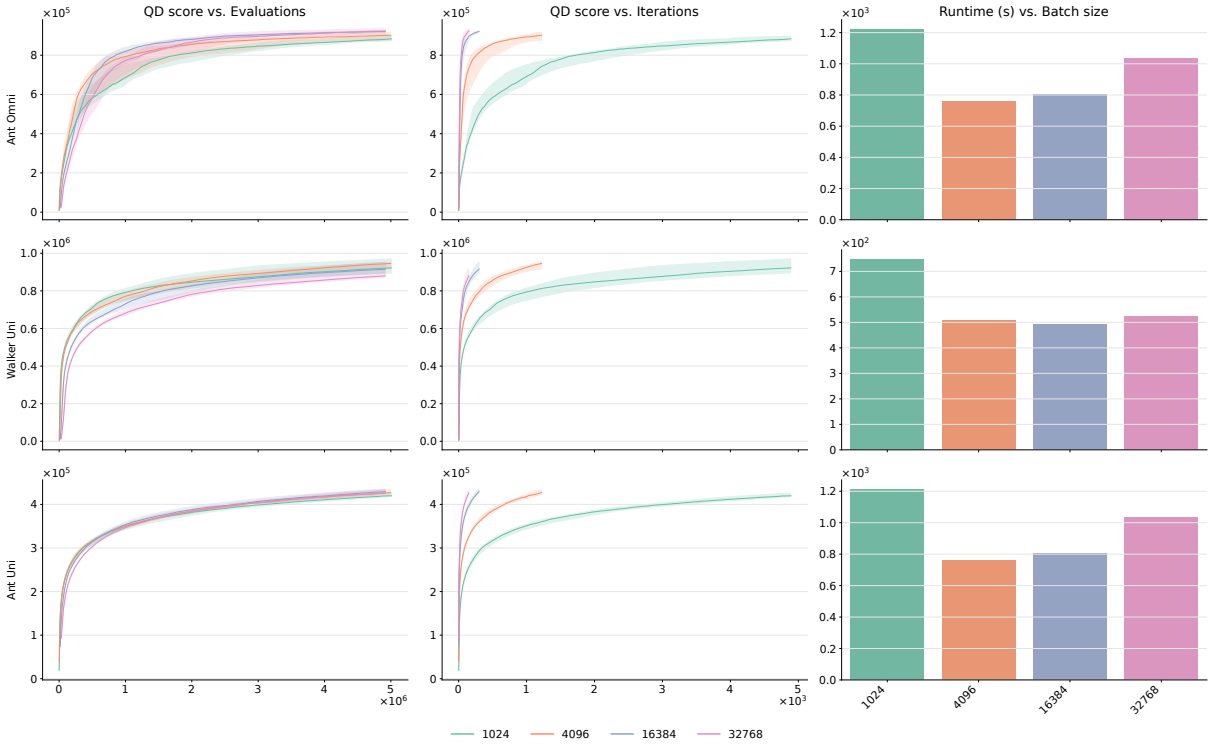


Figure 4.11: The first and second columns show the QD score (Section 4.2.2) of MCPG-ME with respect to the number of evaluations and iterations, respectively, for each batch size in each task. The rightmost column displays MCPG-ME's runtime for each batch size across the tasks.

A key observation from the first column of Figure 4.11 is that the QD scores converge to the same final score after five million evaluations, with the exception of when the $N_{\mathcal{B}} = 32,768$ in the Walker Uni (250) task. These results, with a significance level of $p > 0.05$, indicate that there are no statistically significant differences between the different batch sizes. It is important to note, however, that larger batch sizes exhibit a trend toward being less sample efficient at lower numbers of evaluations, as evidenced by a slight shift to the right in the start of the runs with increasing $N_{\mathcal{B}}$.

The runtime plots show that for all tasks there is a significant decrease in processing time when $N_{\mathcal{B}}$ is reduced from 1024 to 4096, almost halving the time required. Conversely, for larger $N_{\mathcal{B}}$, there is a slight trend towards increasing processing times. Given that the design of the MCPG-ME that allows each solution processed by the MCPG emitter to be independent of the batch size, the runtime for processing one solution versus multiple solutions in parallel should theoretically be roughly the same. Therefore, the observed runtime trend is likely due to hardware limitations and the high memory requirements of the MCPG-ME algorithm, primarily stemming from the need for each parent to sample and process a full episode from the buffer during each update step. When parallelisation is not supported by the hardware, processing defaults to a sequential approach. Consequently, at batch sizes exceeding the hardware's parallelisation capacity, runtime becomes highly dependent on the efficiency of sequential processing.

One might expect that increasing the number of iterations for the MCPG-ME algorithm, or '*learning steps*' in the QD framework, would lead to a better final QD score, primarily for two reasons:

1. In each iteration, new solutions are discovered and added to the repertoire which can be selected as stepping stones to form future solutions in subsequent iterations.

2. The information in the trajectory buffer of MCPG-ME becomes increasingly valuable with

each iteration, as progressively higher-performing solutions are evaluated.

However, Figure 4.11 demonstrates that this is not necessarily the case. In fact, excluding the batch size of 32,768 used in the Walker Uni (250), the number of iterations does not significantly affect the performance of the algorithm when larger batch sizes are used. For instance, in tasks such as Ant Uni (250) and Ant Omni (250), using $N_{\mathcal{B}} = 32,768$, which runs only for 154 iterations, performs similarly to using $N_{\mathcal{B}} = 1,024$, which runs for 4,890 iterations.

4.3.3.2 Scalability of PGA-MAP-Elites

To demonstrate that not all RL MAP-Elites based algorithms exhibit the same scalability, and to highlight the synergy between our RL emitter and MAP-Elites in terms of scalability, we conducted the same tests on PGA-MAP-Elites and the results are presented in Figure 4.12.

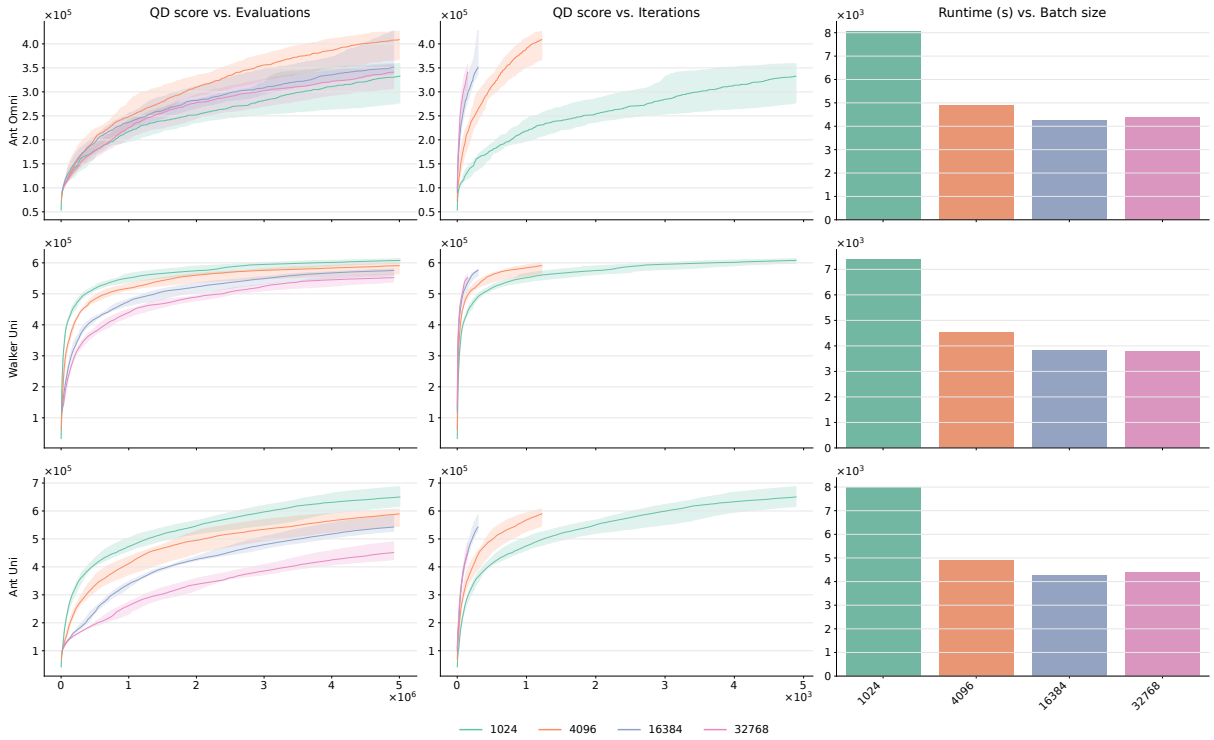


Figure 4.12: The first and second columns show the QD score (Section 4.2.2) of PGA-MAP-Elites with respect to the number of evaluations and iterations, respectively, for each batch size in each task. The rightmost column displays PGA-MAP-Elites' runtime for each batch size across the tasks.

We can observe from Figure 4.12 that PGA-MAP-Elites exhibits patterns opposite to those of MCPG-ME with increasing $N_{\mathcal{B}}$ with respect to both evaluations and iterations in Walker Uni (250) and Ant Uni (250). In particular, with a fixed number of evaluations, the final QD score decreases with larger batch sizes ($p < 0.05$). Furthermore, the lower the number of iterations, which results from increasing $N_{\mathcal{B}}$, the lower the QD score ($p < 0.05$). One possible explanation for the deteriorating performance of PGA-MAP-Elites with increasing batch size and a fixed number of evaluations lies in its 'greediness' components. Unlike our method, the optimisation process in PGA-MAP-Elites' RL emitter relies on a global critic that is trained asynchronously. Additionally, a 'greedy' actor, trained alongside the critic, is evaluated for inclusion in the repertoire during every iteration and frequently contributes very high-performing solutions. These 'greedy' features drive the algorithm's effectiveness, making the number of iterations crucial for

their development. Specifically, the 'greedy' critic improves with each iteration, which in turn enhances its ability to inform the optimisation of solutions drawn from the repertoire. Thus, fewer iterations mean less opportunity for the critic to improve, which affects the overall performance as the critic is directly used for the optimisation of all solutions.

While Figure 4.12 shows variations in final performances for Ant Omni (250) with increasing batch sizes, all comparisons yield $p > 0.1$, indicating no statistically significant differences between the batch sizes. However, due to the high variability in the performance of each batch size, as observed from the wide interquartile ranges in QD scores, this does not necessarily imply that batch size has no effect on the final QD score. More replications are required to draw stronger conclusions. Finally, runtime plots are the only ones that show patterns similar to those of MCPG-ME; however, unlike MCPG-ME, where there is a slight trend toward increasing processing times with larger batch sizes, here we observe that the runtime remains approximately consistent at the runtime of $N_{\mathcal{B}} = 4,096$.

4.3.3.3 Scalability Evaluation

To conclude, MCPG-ME exhibits strong scalability potential, as observed in the third column of Figure 4.11. Increasing the batch size does not compromise the final QD score and simultaneously reduces runtime, especially with adequate hardware support for parallelisation. In contrast, PGA-MAP-Elites shows decreased performance in two of the three tasks with increased batch sizes when the number of evaluations is fixed, indicating weaker scalability.

4.4 Ablation Study

To evaluate the development of the MCPG emitter's objective function, we conduct an ablation study using the same experimental setup as the ones used for assessing sample and time efficiency—each experiment is replicated 10 times using random seeds, across one million evaluations. We focus on unidirectional locomotion tasks with 1000 time-step episodes, specifically Ant Uni (1000), Walker Uni (1000), and Hopper Uni (1000), for two main reasons. First, unidirectional tasks are chosen because the fitness function does not actively discourage diversity in solutions, making the performance of QD tasks highly dependent on this function; this represents a more controlled setup as it minimises confounding variables. Second, selecting episodes with a length of 1000 allows for a more challenging environment, especially due to the limitations of the MC methods, potentially making it easier to distinguish differences in performance.

We compare MCPG-ME against 3 ablations:

- **Ablation 1**: In the final objective function, the importance sampling (IS) ratio-weighted normaliser constant, $Z(\boldsymbol{\theta}) = \sum_{i=1}^{N} \sum_{t=0}^{T-1} r_t^i(\boldsymbol{\theta})$, introduced in Section 3.2.4.2, is replaced with the conventional constant, $Z = N \times T$, representing the number of transitions used to compute the objective function.

- **Ablation 2**: The baseline introduced in Section 3.2.4.2, is removed from the final objective function.

- **Ablation 3**: The IS ratios and the clipping mechanism are removed, and the objective function is reverted to that of REINFORCE. Note that the baseline is retained.

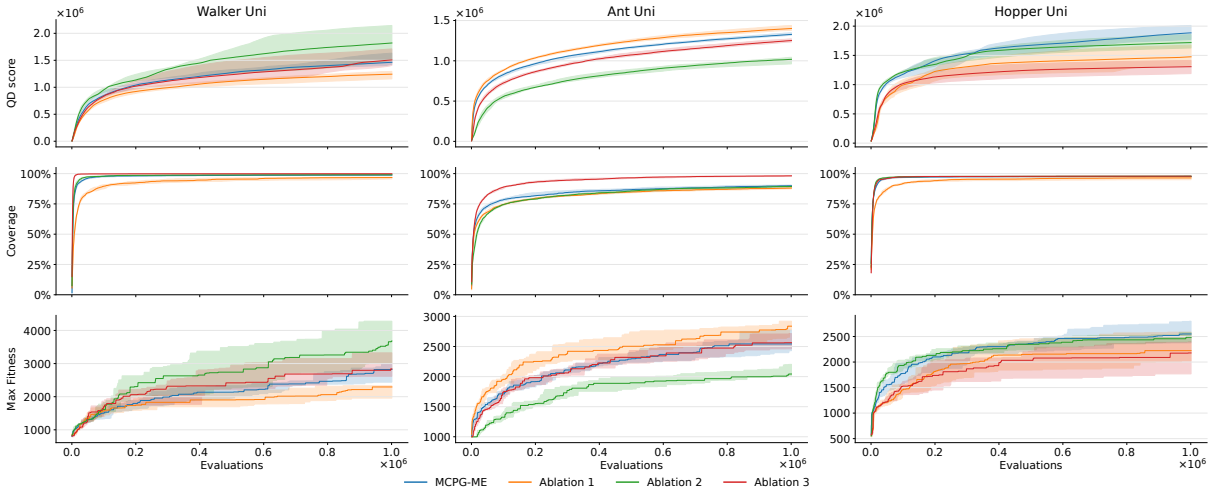


Figure 4.13: QD score, coverage and max fitness (Section 4.2.2) for MCPG-ME and the ablations on the **1000-step tasks**. Each experiment is replicated 10 times with random seeds. The solid line is the median and the shaded area represents the first and third quartiles.

The ablation results presented in Figure 4.13 show that the QD score ranking of all MCPG-ME variants is highly dependent on the task. However, based on the Mann-Whitney U test, comparisons of the two highest-performing ablations against MCPG-ME in the Walker Uni and Hopper Uni tasks have a significance level of $p > 0.05$. Therefore, no strong conclusions can be drawn from these results, other than that these variants perform similarly to MCPG-ME in terms of QD score in these tasks. Given the significant learning instability and frequent policy collapse observed during experiments with the REINFORCE algorithm when used for off-policy learning, it is noteworthy that Ablation 3 achieves a score competitive with MCPG-ME in Walker Uni., while the learning being purely off-policy. Moreover, Ablation 3 achieves significantly higher coverage than MCPG-ME in Ant Uni ($p < 0.005$), reaching almost 100%. This further supports the hypothesis presented in Section 4.3.1 , which posits that learning instabilities introduced by large variances in the estimation of the objective function embrace diversity, thereby aiding in coverage. However, despite this high coverage, Ablation 3 achieves the second lowest QD score, which implies that the solutions collected in the repertoire are not of high fitness. This observation aligns closely with both theoretical expectations and our empirical results, suggesting that in pure RL, the REINFORCE loss does not perform well with off-policy learning. Finally, the fact that MCPG-ME performs similarly to Ablation 2 in both Walker Uni and Hopper Uni tasks raises questions about the effectiveness of using a time-step dependent baseline on tasks where termination of the episode can easier happen.

To conclude, while MCPG-ME does not achieve the highest QD score in all tasks evaluated, it is the only variant that consistently ranks first or second. Moreover, all other variants achieve the lowest QD score in at least one of the three tasks. Therefore, these results justify the design choices made for the objective of our MCPG emitter.

Chapter 5

Conclusions and Future Work

In summary, this work validates the effectiveness of integrating Monte Carlo Policy Gradient (MCPG) methods with MAP-Elites through the development of MCPG-ME. It successfully meets our initial goal of synthesising the rapid execution and scalability of MAP-Elites with the efficient optimisation capabilities of Policy Gradient (PG) methods. Our evaluations across various continuous control locomotion tasks have demonstrated that MCPG-ME matches or sometimes exceeds the performance of the state-of-the-art actor-critic based MAP-Elites algorithms in many of the tasks, while significantly reducing runtime, sometimes being up to nine times faster. Indeed, it achieves a predetermined high performance more quickly than all competitors in most of the tasks, highlighting its notable time efficiency. Furthermore, the algorithm's ability to optimise multiple solutions in parallel—without compromising performance and effectively utilising available hardware resources—directly supports our goal of scalable optimisation.

Despite these strengths, MCPG-ME, like all methods employing PG approaches, is confined to using differentiable function approximators such as neural networks and is limited to fully observable environments that adhere to the constraints of a Markov Decision Process (MDP). Additionally, compared to actor-critic based algorithms, it tends to underperform in more challenging tasks that feature complex fitness functions and larger state and action spaces.

For future work, we aim to explore how a global actor-critic training process affects the scalability of a QD-RL algorithm. Specifically, after conducting further scalability tests on the current actor-critic based MAP-Elites algorithms, we plan to integrate unique critic neural networks for each solution. This approach will help us understand the extent to which scalability limitations in actor-critic methods stem from their global training nature and assess the impact of using individual critics on the sample efficiency of the algorithm. We will modify our current MCPG-ME algorithm by replacing the time-dependent baseline with a critic network that updates concurrently with the policy through a shared objective function. This critic will serve as a state-dependent baseline for its corresponding policy, leveraging the known benefits of such baselines for enhancing learning stability and effectiveness [37]. We anticipate this modification might slightly reduce the speed but enhance the effectiveness of the learning process, while preserve the scalability capabilities of the current algorithm. Regardless of the outcome, this investigation will provide valuable insights into the scalability and efficiency of QD-RL algorithms.

5.1 Ethical Considerations

Our research strictly follows ethical guidelines and is largely free from typical ethical concerns due to its focus. This work is dedicated to developing algorithms under RL and QD frame-

works, employing simulations for testing purposes. This methodology does not involve human or animal subjects, nor does it engage with sensitive personal data, biological materials, or environmental hazards. As such, it bypasses ethical issues related to human embryos, human participants, and data protection, among others. Furthermore, the project strictly uses simulations that do not necessitate or imply any dual use, military application, or potential for misuse that could lead to ethical concerns. Therefore, our development and testing processes remain within the bounds of academic integrity and ethical research practices.

Bibliography

[1] T. Elmqvist, C. Folke, M. Nyström, G. Peterson, J. Bengtsson, B. Walker, and J. Norberg, "Response diversity, ecosystem change, and resilience," pp. 488–494, 11 2003. pages 1

[2] C. Folke, S. Carpenter, B. Walker, M. Scheffer, T. Elmqvist, L. Gunderson, and C. S. Holling, "Regime shifts, resilience, and biodiversity in ecosystem management," pp. 557–581, 2004. pages 1

[3] K. Chatzilygeroudis, A. Cully, V. Vassiliades, and J.-B. Mouret, "Quality-Diversity Optimization: a novel branch of stochastic optimization," 12 2020. [Online]. Available: http://arxiv.org/abs/2012.04322 pages 1

[4] A. Cully and Y. Demiris, "Quality and Diversity Optimization: A Unifying Modular Framework," 5 2017. [Online]. Available: http://arxiv.org/abs/1708.09251 pages 1, 15, 16

[5] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Frontiers Robotics AI*, vol. 3, no. JUL, 7 2016. pages 1, 15, 39

[6] A. Cully, "Autonomous skill discovery with quality-diversity and unsupervised descriptors," in *GECCO 2019 - Proceedings of the 2019 Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, Inc, 7 2019, pp. 81–89. pages 1

[7] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "First return, then explore," *Nature*, vol. 590, no. 7847, pp. 580–586, 2 2021. pages 1

[8] P. L. Lanzi, ACM Digital Library., and Association for Computing Machinery. SIGEVO., *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM, 2011. pages 1

[9] A. Cully, J. Clune, D. Tarapore, and J. B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, 5 2015. pages 1, 16

[10] V. Vassiliades, K. Chatzilygeroudis, and J.-B. Mouret, "Using Centroidal Voronoi Tessellations to Scale Up the Multi-dimensional Archive of Phenotypic Elites Algorithm," 10 2016. [Online]. Available: http://arxiv.org/abs/1610.05729 pages 1, 16

[11] D. Tarapore, J. Clune, A. Cully, and J.-B. Mouret, "How Do Different Encodings Influence the Performance of the MAP-Elites Algorithm?" 2016. [Online]. Available: http://dx.doi.org/10.1145/2908812.2908875 pages 1, 16

[12] F. Chalumeau, T. Pierrot, V. Macé, A. Flajolet, K. Beguir, A. Cully, and N. Perrin-Gilbert, "Assessing Quality-Diversity Neuro-Evolution Algorithms Performance in Hard Exploration Problems," 11 2022. [Online]. Available: http://arxiv.org/abs/2211.13742 pages 1

[13] T. Pierrot, V. Macé, F. Chalumeau, A. Flajolet, G. Cideron, K. Beguir, A. Cully, O. Sigaud, and N. Perrin-Gilbert, "Diversity policy gradient for sample efficient quality-diversity optimization," in *GECCO 2022 - Proceedings of the 2022 Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, Inc, 7 2022, pp. 1075–1083. pages 1, 21

[14] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," 4 2015. [Online]. Available: http://arxiv.org/abs/1504.04909 pages 1, 16, 17, 19, 38

[15] F. Chicano and ACM Special Interest Group on Genetic and Evolutionary Computation, *Proceedings of the Genetic and Evolutionary Computation Conference*. pages 1, 19

[16] M. C. Fontaine, J. Togelius, S. Nikolaidis, and A. K. Hoover, "Covariance matrix adaptation for the rapid illumination of behavior space," in *GECCO 2020 - Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, 6 2020, pp. 94–102. pages 1, 19

[17] M. Flageat and A. Cully, "Fast and stable MAP-Elites in noisy domains using deep grids," 6 2020. [Online]. Available: http://arxiv.org/abs/2006.14253http://dx.doi.org/10.1162/isal_a_00316 pages 1

[18] J. Gomes, S. M. Oliveira, and A. L. Christensen, "An approach to evolve and exploit repertoires of general robot behaviours," *Swarm and Evolutionary Computation*, vol. 43, pp. 265–283, 12 2018. pages 1

[19] A. Cully and Y. Demiris, "Hierarchical behavioral repertoires with unsupervised descriptors," in *GECCO 2018 - Proceedings of the 2018 Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, Inc, 7 2018, pp. 69–76. pages 1

[20] V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," Tech. Rep., 2016. pages 2

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," 12 2013. [Online]. Available: http://arxiv.org/abs/1312.5602 pages 2

[22] R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," Tech. Rep., 1992. pages 2

[23] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," Tech. Rep. pages 2

[24] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," 1 2018. [Online]. Available: http://arxiv.org/abs/1801.01290 pages 2

[25] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 9 2015. [Online]. Available: http://arxiv.org/abs/1509.02971 pages 2

[26] D. Silver, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," Tech. Rep. pages 2

[27] O. Nilsson and A. Cully, "Policy Gradient Assisted MAP-Elites; Policy Gradient Assisted MAP-Elites," 2021. [Online]. Available: https://github.com/ollenilsson19/QDgym pages 2, 19, 20, 38

[28] M. F. A. C. Maxence Faldor, Félix Chalumeau, "Synergizing quality-diversity with descriptor-conditioned reinforcement learning," 2023. [Online]. Available: https://arxiv.org/pdf/2401.08632 pages 2, 20, 38

[29] M. Faldor, F. Chalumeau, M. Flageat, and A. Cully, "MAP-Elites with Descriptor-Conditioned Gradients and Archive Distillation into a Single Policy," 3 2023. [Online]. Available: http://arxiv.org/abs/2303.03832 pages 2, 20

[30] Y. Tang, Y. Tian, and D. Ha, "EvoJAX: Hardware-Accelerated Neuroevolution," in *GECCO 2022 Companion - Proceedings of the 2022 Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, Inc, 7 2022, pp. 308–311. pages 2

[31] F. Chalumeau, B. Lim, R. Boige, M. Allard, L. Grillotti, M. Flageat, V. Macé, G. Richard, A. Flajolet, T. Pierrot, and A. Cully, "QDax: A Library for Quality-Diversity and Population-based Algorithms with Hardware Acceleration," Tech. Rep., 2024. [Online]. Available: https://pypi.org/project/qdax/ pages 2, 36

[32] R. T. Lange, "evosax: JAX-based Evolution Strategies," 12 2022. [Online]. Available: http://arxiv.org/abs/2212.04180 pages 2

[33] B. Lim, M. Allard, L. Grillotti, and A. Cully, "Accelerated Quality-Diversity through Massive Parallelism," 2 2022. [Online]. Available: http://arxiv.org/abs/2202.01258 pages 2, 45

[34] M. Flageat, B. Lim, and A. Cully, "Enhancing MAP-Elites with Multiple Parallel Evolution Strategies," 3 2023. [Online]. Available: http://arxiv.org/abs/2303.06137http://dx.doi.org/10.1145/3638529.3654089 pages 2, 21, 22, 38

[35] C. Colas, V. Madhavan, J. Huizinga, and J. Clune, "Scaling MAP-Elites to deep neuroevolution," in *GECCO 2020 - Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, 6 2020, pp. 67–75. pages 2, 21

[36] Z. Michalewicz, *Evolution Strategies and Other Methods*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 127–138. [Online]. Available: https://doi.org/10.1007/978-3-662-02830-8_9 pages 2

[37] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html pages 4, 12, 50

[38] T. Jie and P. Abbeel, "On a connection between importance sampling and the likelihood ratio policy gradient," in *Advances in Neural Information Processing Systems*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds., vol. 23. Curran Associates, Inc., 2010. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2010/file/35cf8659cfcb13224cbd47863a34fc58-Paper.pdf pages 10

[39] S. Bhatnagar, R. Sutton, M. Ghavamzadeh, M. Lee, and R. S. Sutton, "Natural Actor-Critic Algorithms," *Automatica*, vol. 45, no. 11, p. 10, 2009. [Online]. Available: https://inria.hal.science/hal-00840470 pages 12

[40] J. Yi and X. Liu, "Deep reinforcement learning for intelligent penetration testing path design," *Applied Sciences*, vol. 13, no. 16, 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/16/9467 pages 13

[41] L. Bottou, *Stochastic Gradient Descent Tricks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 421–436. [Online]. Available: https://doi.org/10.1007/978-3-642-35289-8_25 pages 13

[42] C. Darwin, *On the origin of species by means of natural selection, or preservation of favoured races in the struggle for life*. London: John Murray, 1859, with a half-title, binder's directions, and a 32-page list of works published by John Murray. - Colophon varies for the list. - With MS. inscription: H.M.Barton; Reproduction of original in British Library, London.; NSTC 2D3209.; Microfiche. Cambridge : Chadwyck-Healey Ltd., 1990. 6 fiches; 11x15 cm. The Nineteenth Century: General Collection: Science; Pos: Fiche N.1.1.4328. [Online]. Available: https://search.library.wisc.edu/catalog/9934839413602122 pages 14

[43] W. Freeman and Company, "Natural selection in nature," 2010. [Online]. Available: https://www.pinterest.co.uk/pin/1407443628621614/ pages 14

[44] R. G. Lewis and B. Simpson, *Genetics, Autosomal Dominant*. Treasure Island (FL): StatPearls Publishing, 2024, copyright © 2024, StatPearls Publishing LLC. Disclosure: Rueben Lewis declares no relevant financial relationships with ineligible companies. Disclosure: Brittany Simpson declares no relevant financial relationships with ineligible companies. PMID: 32491444. [Online]. Available: https://search.library.wisc.edu/catalog/9934839413602122 pages 14

[45] D. C. Antoine, "Evolutionary algorithms," 2023, introduction to Machine Learning Lecture 7. pages 15

[46] A. Gaier, A. Asteroth, and J. B. Mouretz, "Aerodynamic design exploration through surrogate-assisted illumination," in *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 2017*. American Institute of Aeronautics and Astronautics Inc, AIAA, 2017. pages 16

[47] A. Alvarez, S. Dahlskog, J. Font, and J. Togelius, "IEEE TRANSACTION ON GAMES, VOL. XX, NO. X, MONTH XXXX 1 Interactive Constrained MAP-Elites: Analysis and Evaluation of the Expressiveness of the Feature Dimensions," Tech. Rep. [Online]. Available: https://github.com/mau-games/eddy pages 16

[48] University of London. Queen Mary, IEEE Computational Intelligence Society, and Institute of Electrical and Electronics Engineers, *IEEE Conference on Games 2019 : London, United Kingdom, 20-23 August 2019*. pages 16

[49] J. C. J.-B Mouret, "Illuminating search spaces by mapping elites." 2015. [Online]. Available: https://members.loria.fr/JBMouret/qd.html pages 17

[50] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber, "Natural evolution strategies," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 2008, pp. 3381–3387. pages 18

[51] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution Strategies as a Scalable Alternative to Reinforcement Learning," 3 2017. [Online]. Available: http://arxiv.org/abs/1703.03864 pages 18

[52] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," 2 2018. [Online]. Available: http://arxiv.org/abs/1802.09477 pages 19

[53] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347 pages 28, 29

[54] A. Doerr, M. Volpp, M. Toussaint, S. Trimpe, and C. Daniel, "Trajectory-based off-policy deep reinforcement learning," 2019. [Online]. Available: https://arxiv.org/abs/1905.05710 pages 34

[55] L. Peshkin and C. R. Shelton, "Learning from scarce experience," *CoRR*, vol. cs.AI/0204043, 2002. [Online]. Available: https://arxiv.org/abs/cs/0204043 pages 34

[56] N. Meuleau, L. Peshkin, L. P. Kaelbling, and K.-E. Kim, "Off-policy policy search," 2007. [Online]. Available: https://api.semanticscholar.org/CorpusID:15341735 pages 34

[57] D. Precup, R. Sutton, and S. Singh, "Eligibility traces for off-policy policy evaluation," *Computer Science Department Faculty Publication Series*, 06 2000. pages 34

[58] C. R. Shelton, "Policy improvement for pomdps using normalized importance sampling," *CoRR*, vol. abs/1301.2310, 2013. [Online]. Available: http://arxiv.org/abs/1301.2310 pages 34

[59] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/google/jax pages 36

[60] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, "Brax - a differentiable physics engine for large scale rigid body simulation," 2021. [Online]. Available: http://github.com/google/brax pages 37

[61] M. Flageat, B. Lim, L. Grillotti, M. Allard, S. C. Smith, and A. Cully, "Benchmarking quality-diversity algorithms on neuroevolution for reinforcement learning," 2022. [Online]. Available: https://arxiv.org/abs/2211.02193 pages 39

Appendices

.1 Hyperparameters of Baseline Algorithms

Table 1: DCG-MAP-Elites-AI hyperparameters

Parameter	**Value**
Number of centroids	1024
Total batch size b	512
GA batch size b_{GA}	256
PG batch size b_{PG}	128
AI batch size b_{AI}	128
Policy networks	[64, 64, $\|\mathcal{A}\|$]
GA variation param. 1 σ_1	0.005
GA variation param. 2 σ_2	0.05
Actor network	[128, 128, $\|\mathcal{A}\|$]
Critic network	[256, 256, 1]
TD3 batch size N	100
Critic training steps n	3000
PG training steps m	150
Policy learning rate	5×10^{-3}
Actor learning rate	3×10^{-4}
Critic learning rate	3×10^{-4}
Replay buffer size	10^6
Discount factor γ	0.99
Actor delay Δ	2
Target update rate	0.005
Smoothing noise var. σ	0.2
Smoothing noise clip	0.5
Length scale l	0.1

Table 2: PGA-MAP-Elites hyperparameters

Parameter	Value
Number of centroids	1024
Total batch size b	512
GA batch size b_{GA}	256
PG batch size b_{PG}	255
AI batch size b_{AI}	1
Policy networks	[64, 64, $\|\mathcal{A}\|$]
GA variation param. 1 σ_1	0.005
GA variation param. 2 σ_2	0.05
Actor network	[128, 128, $\|\mathcal{A}\|$]
Critic network	[256, 256, 1]
TD3 batch size N	100
Critic training steps n	3000
PG training steps m	150
Policy learning rate	5×10^{-3}
Actor learning rate	3×10^{-4}
Critic learning rate	3×10^{-4}
Replay buffer size	10^6
Discount factor γ	0.99
Actor delay Δ	2
Target update rate	0.005
Smoothing noise var. σ	0.2
Smoothing noise clip	0.5

Table 3: MAP-Elites hyperparameters

Parameter	Value
Number of centroids	1024
Total batch size b	512
GA batch size b_{GA}	512
Policy networks	[64, 64, $\|\mathcal{A}\|$]
GA variation param. 1 σ_1	0.005
GA variation param. 2 σ_2	0.05

Table 4: MEMES hyperparameters

Parameter	Value
Number of centroids	1024
Total batch size b	32
Samples sigma	0.02
Num. Samples	512
Learning rate	0.01
Num. nearest neighbours	10
l_2 normalisation coefficient	0
Reset budget S_{max}	32
Proportion explore p_{explore}	0.5
Fifo size	50000
Policy networks	[64, 64, $\|\mathcal{A}\|$]