Accessible Quality Diversity Optimization

by

Bryon Tjanaka

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

December 2025

Acknowledgements

Doing a Ph.D. is hard. Fortunately, it is never done alone. Like many things in life, it is easier — and frankly more fun — when done with friends. To that end, I would like to thank the many folks who accompanied and guided me on this journey.

As an undergraduate at UC Irvine, Gregoria Barazandeh introduced me to research in the Undergraduate Research Opportunities Program. From there, I began my first research project on security robots and privacy concerns with Caesar Sereseres, and then worked with David Mobley and Jessica Maat on force fields for molecular simulations. One summer, I interned with Mert Terzihan and Gokhan Mergen on a research engineering team in Google Ads, and in my final year, I worked with Roy Fox on reinforcement learning. Along the way, I joined UCI's ACM chapter, meeting folks like Karthik Gajulapalli and Arne Philipeit. I am grateful to all these folks for showing me what it meant to do research and inspiring me to pursue my Ph.D. They taught me that research, like flying, is about learning to throw yourself at the ground and miss.

Throughout the course of my Ph.D., I have had the most amazing colleagues in the ICAROS Lab. Thank you Matt Fontaine for introducing me to quality diversity (QD) and for your mentorship throughout the years. Thank you Varun Bhatt for always staying on top of things and keeping the lab hardware from burning down. Thank you Heramb Nemlekar for being an awesome social chair. Thank you Nathan Dennler for your conversations about dance. Thank you Sophie Hsu for your work on BALLU and your many supportive conversations. Thank you David Lee and Henry Chen for being amazing mentees and for your work on pyribs and discount models, respectively. Thank you Aaquib Tabrez for being the world's best boss during your year as a postdoc here. Thank you Shihan Zhao for your enthusiasm about pyribs. Thank you Saeed Hedayatian for our many late-night conversations on QD; I think you and Shihan are going to do great in your Ph.D.s!

On the other hand, the folks in the USC Graduate School and the Department of Computer Science have made my life astronomically easier by taking care of the mountains of paperwork and other administrative duties required to support Ph.D. students like myself. In particular, thank you Meredith Drake Reitan and Kate Tegmeyer from the Graduate School, and thank you Lizsl De Leon and Rita Wiraatmadja from the Department of Computer Science.

One of the pleasant aspects of working on QD is that the community is fairly tight-knit, and everyone knows everybody else. I still remember how when I attended my first conference (GECCO 2022), everyone could fit in a single room. Naturally, the community has grown since then, but it still feels like I know most of the folks. To that end, thank you to the QD community for being welcoming and encouraging. In particular, I appreciate the community's support of the pyribs library; the many users of the library have inspired me to improve and expand it throughout my Ph.D. I would also like to thank Thomas Pierrot, who invited me to intern at InstaDeep in 2023.

Beyond the Ph.D., I wish to thank the people who have given color to the world outside my studies. Thank you to my family: my parents Willy and Hun-Yi, my brother Dylon, and Estelle Kao. Thank you to Sebastian Ojeda and the Viterbi Graduate Student Association for welcoming me into an amazing community during my first few years at USC. Thank you to the USC Ballroom and Latin Dance Team for helping me grow my passion for ballroom dance: Kristina Andreyeva, Cerllin Chill, Claire Katen, Madelyn Officer, Helen Sun, Alexey Tregubov, Bryn Tronco, Metehan Aksay, and countless others. Thank you Lea Stith, and thank you Tanya Estrina and the MIT Ballroom Dance Team, for keeping me company during my time interning in Boston.

Finally, thank you to my advisor, Stefanos Nikolaidis, for your mentorship throughout my Ph.D. I have always been able to count on you for your support, and for that I am forever grateful.

Table of Contents

List of Tables

List of Figures

Abstract

Quality diversity (QD) optimization is a branch of stochastic optimization that aims to search for a diverse, high-performing collection of solutions. For example, given the task of training a humanoid robot to walk, a typical single-objective optimization algorithm would teach the robot to walk in only one way. In contrast, QD would teach the robot to walk in many ways: normally, tip-toeing slowly, running quickly, and even skipping and jumping. Historically, QD algorithms have focused on relatively small problems, limiting their applicability in areas like machine learning. In this dissertation, I propose that QD algorithms can be made more accessible by scaling them to overcome three key bottlenecks. First, I show how to scale QD to high-dimensional solutions, enabling QD to tackle problems in deep reinforcement learning. Second, I show how to scale QD to problems with long evaluation times, making QD more accessible in environment generation. Third, I show how to scale QD to more complex specifications of diversity, enabling a rich array of machine learning problems to be specified as QD problems. Furthermore, to make these methods available to the community, I present pyribs, a software package that implements these methods and many other QD algorithms. By scaling up QD in this manner, these works provide an algorithmic and software toolbox that makes QD accessible for a wide variety of practitioners.

Chapter 1

Introduction

1.1 Motivation

Quality diversity (QD) optimization proposes to find a diverse, high-performing set of solutions to a given problem [190]. For example, consider a humanoid robot that has the objective of learning to move forward. A traditional single-objective optimization algorithm would train the humanoid to move forward as quickly as possible, e.g., by running. However, there are many ways to move forward that each have their own benefits. For example, the humanoid could be trained to tip-toe forward slowly, which might be useful in quiet areas like libraries. Or it could learn to walk normally, so that it can move along as part of crowds. It can even learn to hop along, which is perhaps useful on rugged terrain. While the single-objective optimization algorithm would find only the fastest way to move forward, a QD algorithm would seek to learn all of these ways to move forward. A single execution of a QD algorithm would produce a set of solutions that enable the humanoid robot to tip-toe, walk, hop, run, and so forth. This paradigm of searching for diverse, high-performing solutions to a problem has proven powerful in many domains, such as reinforcement learning [175, 42, 239, 186, 240, 44, 14], robot manipulation [165, 164], human-robot interaction [71, 76, 74], video game level generation [75, 61], agent testing [19], generative modeling [73], urban planning [86], design [83], internet congestion control [64], and drug discovery [251].

Formally, a QD problem considers an *objective* function $f : \mathbb{R}^n \to \mathbb{R}$ and *measure* function $m : \mathbb{R}^n \to \mathbb{R}^k$, where $\mathbb{R}^n$ is the n-dimensional *solution space* and $\mathbb{R}^k$ is the k-dimensional *measure space*. The goal of a QD algorithm is to find a set of solutions that elicit diverse outputs of the measure function while maximizing the objective. In the earlier example, the humanoid robot maximizes the objective f of moving forward, while the measure function m represents

Figure 1.1: General layout of a QD algorithm.

the different ways to move forward, like tip-toeing and running. Each way of moving forward is represented by an n-dimensional solution, e.g., the solution could be the parameters of a controller for the humanoid.

A QD algorithm typically proceeds in three steps, as shown in Fig. 1.1. First, the algorithm **generates** new candidate solutions $\boldsymbol{\theta}_i$. Next, it **evaluates** the solutions to determine their objective values $f(\boldsymbol{\theta}_i)$ and measure values $\boldsymbol{m}(\boldsymbol{\theta}_i)$. Finally, the algorithm adds the solutions to an *archive*, i.e., it **updates** the archive. To elaborate, the archive stores the collection of diverse, high-performing solutions found by the QD algorithm. After running for some number of iterations, the QD algorithm outputs the archive to the user.

In this dissertation, I address limitations that QD algorithms have encountered in each of these steps:

1. **High-Dimensional Solutions:** When *generating solutions*, QD algorithms typically assume the solutions have relatively few parameters (i.e., $n < 100$). However, many problems in machine learning require high-dimensional solutions, e.g., in deep reinforcement learning, policies/controllers frequently require tens of thousands of parameters. Such high dimensionalities make it challenging to generate solutions that achieve adequate diversity and performance.

2. **Long Evaluation Times:** When *evaluating solutions*, QD algorithms have assumed access to quick evaluations. However, in domains like environment generation, evaluations can take seconds or even minutes, making it intractable to run a sufficient number of evaluations.

3. **High-Dimensional Measures:** When *updating the archive*, QD algorithms have assumed low-dimensional representations of diversity, specifically, a low-dimensional measure space $\mathbb{R}^k$. This occurs because high-dimensional measures ($k \gg 10$) create a host of challenges. However, high-dimensional measures offer the opportunity to more flexibly specify the diversity of solutions needed in a QD problem.

Collectively, these limitations have restricted the applicability of QD algorithms to relatively small problems. Hence, in this dissertation, *I propose that addressing these limitations will enhance the accessibility of QD algorithms, making them applicable in a wide variety of problems*. To that end, I propose methods to address each limitation and show how those methods each enhance an application area for QD. Furthermore, I propose pyribs, a software package that implements an array of QD algorithms, including these advances, greatly improving the accessibility of QD for the community. In the remainder of this chapter, I overview the development of each method.

1.2 Scaling to High-Dimensional Solutions

The majority of QD algorithms operate with low-dimensional solutions, i.e., with solution space dimensionality $n < 100$. However, in fields such as deep reinforcement learning (RL), a policy/controller can easily require tens of thousands of parameters. Even a relatively small policy, consisting of a neural network with two hidden layers, can require 20,000 parameters. This large number of parameters makes it difficult to identify promising search directions and generate high-performing solutions.

To address these issues, I first propose a black-box QD method that scales to high-dimensional solutions. As described in Chapter 2, this method builds on CMA MAE, a state-of-the-art QD algorithm. CMA-MAE is unable to operate in high-dimensional search spaces because it requires $O(n^2)$ runtime to generate a new solution. For $n \geq 1000$, CMA-MAE quickly becomes computationally intractable. Thus, Chapter 2 proposes to replace certain components of CMA-MAE with approximations that reduce the runtime to $O(n)$, or $O(dn)$ with a small constant d.

While the black-box method described in Chapter 2 can tackle a wide range of problems, it does not fully leverage the information available in the RL formulation, which prevents it from achieving maximal performance. RL is typically formulated as a Markov decision process [234], and the objective in an RL problem is the summation of rewards received at each timestep of an agent's operation. Deep RL algorithms [81, 103] leverage this structure to efficiently train high-performing policies by creating a surrogate objective and performing gradient ascent on that objective. In contrast, Chapter 2 assumes the objective is a black box, i.e., that no information is available about it beyond the output.

To search more effectively, Chapter 3 and Chapter 4 integrate deep RL algorithms with QD algorithms. Specifically, these methods build on *differentiable quality diversity (DQD)* [73], a paradigm of QD that assumes the objective and measures are first-order differentiable. Since the objective in RL is not differentiable, DQD cannot directly be applied.

To overcome this issue, Chapter 3 introduces the idea of using existing RL algorithms to approximate gradients for existing DQD algorithms. Chapter 4 takes this idea a step further by deeply integrating PPO [216] (a deep RL algorithm) into DQD, resulting in state-of-the-art performance among QD algorithms for deep RL. Together, these methods form a toolbox that makes QD algorithms accessible to deep RL practitioners.

1.3 Scaling to Long Evaluation Times

For various reasons, QD algorithms are rather sample-inefficient, i.e., they must sample and evaluate a large number of solutions to create the final archive. When evaluating a solution is computationally cheap, this approach is feasible, and millions of evaluations can be run. However, some problems require long, computationally expensive evaluations. For example, in environment generation, a QD algorithm searches for environments that elicit diverse behaviors of agents. A single evaluation requires running one or more agents in an environment found by the QD algorithm. Furthermore, the agents may need to be run multiple times due to stochasticity in the evaluations. Thus, it is intractable to run millions of evaluations, and algorithms only have a budget of several thousand evaluations.

To that end, Chapter 5 proposes to integrate surrogate models into existing QD algorithms to enhance their sample efficiency. A surrogate model provides a computationally cheap approximation of the objective and measure functions. By carefully leveraging the surrogate model, a QD algorithm can predict which solutions are likely to have high objectives and diverse measures. For example, if the surrogate model predicts that a solution will have a low objective value, the QD algorithm can avoid evaluating it, thus reducing the total number of evaluations needed. Chapter 5 shows how this technique enhances a QD algorithm's ability to generate environments for testing agents.

1.4 Scaling to High-Dimensional Measures

QD algorithms typically assume the measure space is fairly low-dimensional, i.e., the measure space dimensionality k is less than 10. In fact, many QD works only consider two measures, i.e., $k = 2$. However, high-dimensional measures offer the opportunity for more complex specifications of diversity. For instance, the variety of humanoid robot gaits may require more than just two measures to describe. Chapter 6 studies the challenges posed by high-dimensional measures and presents an algorithm that addresses these challenges. This algorithm primarily addresses *distortion*, which is a

problem where many solutions have similar measures, making it difficult for the QD algorithm to distinguish between them. Ultimately, Chapter 6 shows how this algorithm makes QD more accessible to the broader machine learning community by enabling specifying measures with datasets of values, rather than hand-designing measure functions.

1.5 Pyribs

While the aforementioned methods all improve the applicability of QD algorithms to some extent, it is difficult for them to have impact without a reliable, easy-to-use software implementation. To make these methods truly accessible to QD practitioners, Chapter 7 presents pyribs, a "bare-bones" Python library for QD. Pyribs implements both the aforementioned methods and a large set of QD algorithms developed by the community. Pyribs divides QD algorithms into modular components with a conceptual framework called RIBS. The software itself is implemented in Python and is supported by extensive documentation and tutorials. Since its inception in 2021, pyribs has grown to support the research of dozens of groups across academia and industry worldwide. As of writing, it has been applied to image generation [73, 72], video game level generation [61], environment generation [19], reinforcement learning [240, 239], hyperparameter optimization [211], architecture design [85], internet congestion control [64], chicken farming [166], and model rocketry [212].

1.6 Conclusions and Future Work

This dissertation proposes to make QD more accessible by developing methods that scale QD to overcome three common bottlenecks: high-dimensional solutions, long evaluation times, and high-dimensional measures. This dissertation describes each of these methods and shows how they have facilitated new applications of QD. There are a wide variety of problems that are best solved by finding a diversity of solutions rather than a single solution. I believe future work should focus on identifying such problems and solving them with the algorithmic and software toolbox presented in this dissertation.

Chapter 2

Scaling Black-Box Quality Diversity to High-Dimensional Controllers

2.1 Introduction

By generating a diverse collection of controllers, we can endow a robot with a variety of useful behaviors. For example, one popular approach in robotic locomotion has been to train a collection of neural network controllers to enable a walking robot to adapt to damage [49, 42, 239, 175]. The controllers differ by how often each foot contacts the ground, such that if a foot is damaged, the robot can select a controller that does not rely on that foot.

Searching for diverse controllers may be viewed as a quality diversity (QD) optimization problem [190]. In QD, the goal is to find solutions ϕ that are diverse with respect to one or more measure functions $m_i(\phi)$ while maximizing an objective function $f(\phi)$. In the locomotion example presented, we search for neural network controller policies π_ϕ parameterized by ϕ. Each controller should satisfy a unique output of the measure function by using its feet in a different manner from the other controllers, while optimizing the objective by walking forward quickly.

A QD algorithm must balance two aspects given a limited compute budget: *exploring* measure space and *optimizing* the objective. In our locomotion example, exploration finds new controllers that use the robot's feet a different amount, and optimization makes existing controllers walk faster.

Prior algorithms [239, 175] seem to strike a balance between these two aspects of QD, leading to state-of-the-art results. However, these algorithms have practical limitations due to their dependence on deep reinforcement learning (RL) methods. Namely, they must perform time-consuming training of a neural network and have many hyperparameters.

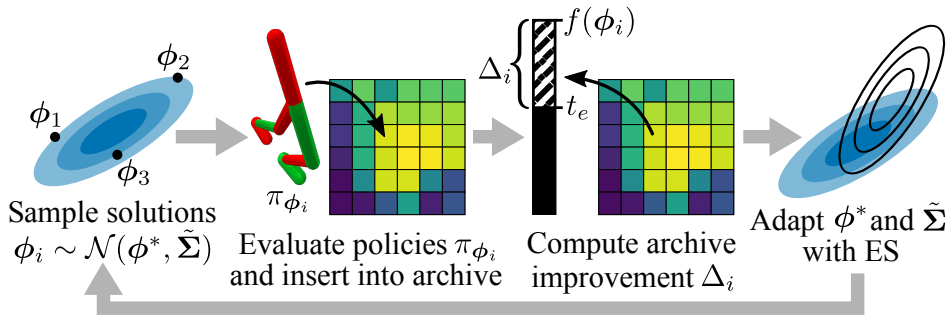


Figure 2.1: We propose variants of the CMA-MAE algorithm that scale to high-dimensional controllers. The variants maintain a Gaussian search distribution with mean ϕ^* and approximate covariance matrix $\tilde{\Sigma}$. Solutions ϕ_i sampled from the Gaussian are evaluated and inserted into an archive, where they generate improvement feedback Δ_i based on their objective value $f(\phi_i)$ and a threshold t_e that each archive cell maintains. Finally, the Gaussian is updated with an evolution strategy (ES). Our variants differ from CMA-MAE by incorporating scalable ESs, as the CMA-ES used in CMA-MAE has $\Theta(n^2)$ time complexity per sampled solution.

Recent work [205] suggests evolution strategies (ES) as a compelling alternative to deep RL methods when optimizing a single controller. Compared with deep RL, ESs do not require network training, and ESs such as the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [108] are designed to have almost no hyperparameters. Given these benefits, prior work [42, 239] has developed QD algorithms based on ESs, but these methods have not yet matched the performance of deep RL-based QD methods.

On the other hand, the recently proposed ES-based Covariance Matrix Adaptation MAP-Annealing (CMA-MAE) algorithm [72] has proven adept at trading off the exploration and objective optimization aspects of QD. Tuning a single hyperparameter $\alpha \in [0, 1]$ in CMA-MAE enables blending these two aspects, yielding state-of-the-art performance on QD benchmarks. We hypothesize that CMA-MAE's ability to balance this tradeoff would enable it to excel in training neural network controllers for robotic locomotion tasks.

While CMA-MAE excels at moderate-dimensional domains, it is intractable for modern neural network controllers because such controllers are *high-dimensional*, i.e., they have thousands or even millions of parameters. Internally, CMA-MAE guides the QD search with one or more CMA-ES instances [108]. Since CMA-ES's time complexity is quadratic in the number of parameters, it cannot scale to such controllers.

CMA-ES's complexity arises from how it models the search distribution with a Gaussian that has a full rank $n \times n$ covariance matrix. However, by replacing this full matrix with sparse approximations, prior work [247] creates variants of CMA-ES that scale to high-dimensional problems.

Our key insight is that we can scale CMA-MAE to high-dimensional controllers by adopting such approximations in its CMA-ES components. Following this insight, we propose three scalable CMA-MAE variants (Sec. 2.3). To understand their performance and runtime properties, we study these variants on optimization benchmarks (Sec. 2.4). Next, we evaluate the variants on robotic locomotion tasks (Sec. 2.5). We show that our variants are the highest-performing QD methods based solely on ES. Furthermore, they are comparable to or exceed the state-of-the-art deep RL-based QD method PGA-ME [175] on three of four tasks, while inheriting the aforementioned practical benefits of ES. We are excited about future applications in other domains, such as robotic manipulation [164] and scenario generation [19], and we have open-sourced our variants in the pyribs library [238].

2.2 Background

2.2.1 Formulation

Quality diversity (QD). Drawing from the definition in prior work [73], QD considers an objective function $f(\boldsymbol{\phi})$ and k-dimensional measure function $\boldsymbol{m}(\boldsymbol{\phi})$,* where $\boldsymbol{\phi} \in \mathbb{R}^n$ is an n-dimensional solution. The outputs of $\boldsymbol{m}$ form a k-dimensional *measure space* $\mathcal{X}$. The *QD objective* is to find, for every $\boldsymbol{x} \in \mathcal{X}$, a solution $\boldsymbol{\phi}$ such that $\boldsymbol{m}(\boldsymbol{\phi}) = \boldsymbol{x}$ and $f(\boldsymbol{\phi})$ is maximized. Solving this QD objective would require infinite memory since $\mathcal{X}$ is a continuous space, so algorithms based on MAP-Elites [169] relax the QD objective by discretizing $\mathcal{X}$ into a tesselation $\mathcal{Y}$ of M cells. Then, the QD objective is to maximize the (sum of) objective values of an *archive* $\mathcal{A}$ containing solutions $\boldsymbol{\phi}_{1..M}$, i.e., $\max_{\boldsymbol{\phi}_{1..M}} \sum_{i=1}^{M} f(\boldsymbol{\phi}_i)$. Furthermore, $\boldsymbol{\phi}_{1..M}$ are constrained such that each $\boldsymbol{\phi}_i$ has measures $\boldsymbol{m}(\boldsymbol{\phi}_i)$ corresponding to a unique cell in $\mathcal{Y}$.

Quality diversity reinforcement learning (QD-RL). As defined in prior work [239], QD-RL is a special instance of QD where $\boldsymbol{\phi}$ parameterizes a reinforcement learning (RL) agent's policy $\pi_{\boldsymbol{\phi}}$, and the objective is the agent's expected discounted return in a Markov Decision Process (MDP) [234]. QD-RL also includes a k-dimensional measure function $\boldsymbol{m}(\boldsymbol{\phi})$ that describes the agent's behavior during an episode.

*It is common to define $\boldsymbol{m}(\boldsymbol{\phi})$ via k separate measure functions $m_i(\boldsymbol{\phi})$. Prior work also refers to measure function outputs as behavior descriptors or behavior characteristics.

2.2.2 Large-Scale Evolution Strategies

An evolution strategy (ES) [17] optimizes continuous parameters by adapting a population of solutions such that the population is more likely to attain high performance. A *large-scale* ES scales to high-dimensional search spaces.

OpenAI-ES [205] is one large-scale ES notable for performing well in RL domains. It represents a population with an isotropic Gaussian and updates only the Gaussian's mean by passing approximated gradients to Adam [136].

Several large-scale ESs build on Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [108], an approximate second-order method that achieves state-of-the-art results in black-box optimization [110]. CMA-ES models a distribution of search directions with a Gaussian $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Every iteration, CMA-ES samples λ solutions from this Gaussian and updates it based on rankings of the solutions' performance.

CMA-ES itself does not scale to high dimensions, as it requires $\Theta(n^2)$ space and $\Theta(n^2)$ runtime per sampled solution. The space is due to the $n \times n$ covariance matrix $\boldsymbol{\Sigma}$, while runtime stems from two operations. First, updating $\boldsymbol{\Sigma}$ requires matrix-vector multiplications. Second, since it is easy to sample from the standard Gaussian $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ on a computer, sampling from $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is implemented as:

$$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \sim \boldsymbol{\mu} + \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}) \sim \boldsymbol{\mu} + \boldsymbol{\Sigma}^{\frac{1}{2}} \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{2.1}$$

The *transformation matrix* $\boldsymbol{\Sigma}^{\frac{1}{2}}$ requires an $O(n^3)$ eigendecomposition, which CMA-ES amortizes to $O(n^2)$ per sampled solution by only recomputing $\boldsymbol{\Sigma}^{\frac{1}{2}}$ every $\frac{n}{\lambda}$ iterations.

Multiple variants [247] of CMA-ES scale to high dimensions by replacing the full covariance matrix with an efficient approximation. We incorporate OpenAI-ES and two such variants to scale CMA-MAE to high dimensions.

2.2.3 MAP-Elites

Many QD algorithms, including those in this work, build on Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) [169]. The vanilla version of MAP-Elites divides the measure space into an archive of evenly-sized grid cells. Then, it generates solutions by sampling existing solutions from the archive and applying a genetic operator. These new solutions are inserted into archive cells based on their measures. If they land in the same cell as a previous solution, they replace the solution only if they have a higher objective.

One recent line of work integrates CMA-ES into MAP-Elites to optimize for the QD objective (Sec. 2.2.1). In Covariance Matrix Adaptation MAP-Elites (CMA-ME) [77], CMA-ES directly samples solutions, adapting the search distribution to find solutions that create the greatest archive improvement. CMA-ME runs multiple CMA-ES instances in parallel, each encapsulated in an *emitter* — emitters are QD algorithm components that generate solutions for evaluation [77, 238]. Meanwhile, Covariance Matrix Adaptation MAP-Elites via a Gradient Arborescence (CMA-MEGA) [73] operates in the differentiable quality diversity (DQD) setting, where exact objective and measure gradients are available. Here, instead of sampling solution parameters, CMA-ES branches from a solution point by sampling coefficients that form linear combinations of the objective and measure gradients.

Multiple methods extend MAP-Elites to train neural network controllers, as vanilla MAP-Elites performs poorly in such problems [42, 239, 175]. For instance, CMA-MEGA cannot be applied to QD-RL since it assumes gradients are provided, and such gradients are often unavailable in RL due to non-differentiable environments. Hence, recent work [239] introduces CMA-MEGA variants that instead approximate the gradients. Meanwhile, MAP-Elites with Evolution Strategies (ME-ES) [42] integrates OpenAI-ES to improve the objective value of a solution point or move the point to a new area of the archive. Finally, Policy Gradient Assisted MAP-Elites (PGA-ME) [175] replaces the genetic operator with two operations: (1) gradient ascent, performed with TD3 [81], and (2) crossover, performed with a genetic algorithm [249]. We include these methods as experimental baselines.

2.2.4 CMA-MAE

We extend Covariance Matrix Adaptation MAP-Annealing (CMA-MAE) [72], a method that builds on CMA-ME and achieves state-of-the-art performance on QD benchmarks.

The key difference between CMA-MAE and CMA-ME is a *soft archive* that enables balancing between optimizing the objective and searching for solutions with new measure values. This soft archive records a *threshold* t_e for each cell e. t_e is initialized to a minimum objective min_f. When a solution $\boldsymbol{\phi}$ is inserted into the archive, it is placed into its corresponding cell e if its objective value $f(\boldsymbol{\phi})$ exceeds t_e. Then, t_e is updated via polyak averaging $t_e \leftarrow (1-\alpha)t_e + \alpha f(\boldsymbol{\phi})$, where $\alpha \in [0, 1]$ is the *archive learning rate*. Finally, the insertion returns an *improvement value* $\Delta_i \leftarrow f(\boldsymbol{\phi}) - t_e$, where higher values indicate greater archive improvement. Note that during insertion, the solution's objective value only needs to cross the threshold, rather than the objective value of the solution previously in

the cell. Thus, implementations must track the best solutions separately, as the archive does not always store them like MAP-Elites does.

Like CMA-ME, CMA-MAE maintains one or more emitters. Each emitter contains a CMA-ES instance that directly samples solutions from a Gaussian. By updating the Gaussian based on rankings of the solutions' improvement values, CMA-ES moves the Gaussian towards solutions more likely to generate high improvement.

The archive learning rate α is a key parameter in CMA-MAE. When $\alpha = 0$, the threshold remains at min_f, so the improvement Δ_i always equals the objective $f(\boldsymbol{\phi})$ (minus a constant min_f). This makes CMA-MAE equivalent to CMA-ES, as it optimizes solely for the objective. When $\alpha = 1$, the threshold is equal to the objective value of the solution currently in the cell, which means there is minimal improvement for inserting a solution into a cell with an existing solution. In this case, CMA-MAE is equivalent to CMA-ME, which always prioritizes discovering new solutions in measure space over improving existing solutions. Varying α from 0 to 1 smoothly trades off between these two extremes.

2.3 Scaling CMA-MAE

In CMA-MAE, each emitter uses CMA-ES to update its Gaussian search distribution. Since CMA-ES requires $\Theta(n^2)$ space and $\Theta(n^2)$ runtime per solution (with n the solution dimension), CMA-MAE cannot train high-dimensional neural network controllers. To scale CMA-MAE, we propose three variants that replace CMA-ES with large-scale ESs. These variants differ primarily in the complexity of their covariance matrix approximation, and each variant is named by taking its large-scale ES's name and replacing "ES" with "MAE":

- **LM-MA-MAE** substitutes Limited-Memory Matrix Adaptation ES (LM-MA-ES) [154], a large-scale CMA-ES variant that approximates the transformation matrix $\boldsymbol{\Sigma}^{\frac{1}{2}}$ with $k \ll n$ n-dimensional vectors, each representing a different direction of the search distribution. This rank-k approximation leads to $\Theta(kn)$ complexity.

- **sep-CMA-MAE** substitutes Separable CMA-ES (sep-CMA-ES) [199], a large-scale CMA-ES variant that constrains the covariance matrix $\boldsymbol{\Sigma}$ to be diagonal, yielding $\Theta(n)$ complexity.

- **OpenAI-MAE** substitutes OpenAI-ES [205]. As OpenAI-ES is not a CMA-ES variant, it differs from CMA-ES in several mechanisms, but it nevertheless represents the search distribution with a Gaussian, specifically an

Algorithm 1: CMA-MAE variants. Highlighted lines show differences from CMA-MAE [72].

1 **CMA-MAE variants** $(eval, \boldsymbol{\phi}_0, N, \psi, \lambda, \sigma, \alpha, min_f)$**:**

Input: $eval$ function that rolls out policy $\pi_{\boldsymbol{\phi}}$ and outputs objective $f(\boldsymbol{\phi})$ and measures $\boldsymbol{m}(\boldsymbol{\phi})$, initial solution $\boldsymbol{\phi}_0$, iterations N, number of emitters ψ, batch size λ, initial step size σ, archive learning rate α, minimum objective min_f

Result: Generates $N\psi\lambda$ solutions, storing elites in an archive $\mathcal{A}$

2 Initialize archive $\mathcal{A}$ and threshold $t_e \leftarrow min_f$ for every cell e

3 Initialize ψ emitters, each with mean $\boldsymbol{\phi}^* \leftarrow \boldsymbol{\phi}_0$, covariance matrix approximation $\tilde{\boldsymbol{\Sigma}} \leftarrow \sigma \boldsymbol{I}$, and internal parameters $\boldsymbol{p}$

4 **for** $iter \leftarrow 1..N$ **do**

5 **for** Emitter 1 .. Emitter ψ **do**

6 **for** $i \leftarrow 1..\lambda$ **do**

7 $\boldsymbol{\phi}_i \sim \mathcal{N}(\boldsymbol{\phi}^*, \tilde{\boldsymbol{\Sigma}})$

8 $f(\boldsymbol{\phi}_i), \boldsymbol{m}(\boldsymbol{\phi}_i) \leftarrow eval(\boldsymbol{\phi}_i)$

9 $e \leftarrow$ calculate_archive_cell($\mathcal{A}, \boldsymbol{m}$)

10 $\Delta_i \leftarrow f(\boldsymbol{\phi}_i) - t_e$

11 **if** $f(\boldsymbol{\phi}_i) > t_e$ **then**

12 Replace the solution in cell e of archive $\mathcal{A}$ with $\boldsymbol{\phi}_i$

13 $t_e \leftarrow (1-\alpha)t_e + \alpha f(\boldsymbol{\phi}_i)$

14 Rank $\boldsymbol{\phi}_i$ by Δ_i

15 Adapt $\boldsymbol{\phi}^*, \tilde{\boldsymbol{\Sigma}}, \boldsymbol{p}$ based on improvement ranking Δ_i

16 **if** ES algorithm converges **then**

17 Restart emitter with $\boldsymbol{\phi}^* \leftarrow$ a randomly selected elite in $\mathcal{A}$, $\tilde{\boldsymbol{\Sigma}} \leftarrow \sigma \boldsymbol{I}$, and new internal parameters $\boldsymbol{p}$

isotropic Gaussian with constant covariance $\sigma \boldsymbol{I}$. Though the covariance is constant, vector operations on the solutions still necessitate $\Theta(n)$ complexity.

The listed complexities refer to (1) the space required per emitter, as each emitter maintains its own ES instance, and (2) the runtime required per sampled solution, which is the same regardless of the number of emitters.

Algorithm 1 and Fig. 2.1 show an overview of the variants. Each variant begins by initializing the archive along with each emitter's ES parameters (line 2-3). This step includes initializing the covariance matrix approximation $\tilde{\boldsymbol{\Sigma}}$ in lieu of the full covariance matrix $\boldsymbol{\Sigma}$ used in CMA-MAE. Next, each variant repeatedly queries the emitters for solutions (line 4). Each emitter (line 5) samples λ solutions from the distribution $\mathcal{N}(\boldsymbol{\phi}^*, \tilde{\boldsymbol{\Sigma}})$ (line 6-7). Note that the sampling procedure depends on the approximation employed by the variant. Once sampled, the solutions are evaluated and inserted into the archive if they cross their cell's threshold t_e (line 8-13). Then, $\boldsymbol{\phi}^*$, $\tilde{\boldsymbol{\Sigma}}$, and the ES's parameters are updated based on the solutions' improvement ranking (line 14-15), such that the emitter is more likely to sample solutions with high improvement on the next iteration. Finally, the emitter restarts if the ES converges (line 16-17). We adopt default update and convergence rules from each ES.

2.4 Optimization Benchmarks

Replacing CMA-ES with large-scale ESs in our CMA-MAE variants raises two questions: (1) Since our variants model the search distribution with an approximate Gaussian rather than a full Gaussian, how do they perform relative to each other and relative to CMA-MAE? (2) In practice, are the variants faster than CMA-MAE? While our goal is to train neural network controllers for robotic locomotion, it is impractical to answer these questions in that domain, since CMA-MAE's quadratic complexity prevents it from training high-dimensional controllers. Thus, we first study the variants on lower-dimensional benchmarks.

2.4.1 Experimental Setup

Domains. We consider three QD benchmarks: (1) In *sphere linear projection* [77], the objective is the sphere function $f(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2$, and the measure function linearly projects solutions into a 2D space. (2) *Arm repertoire* [249] considers a planar robotic arm with n equally-sized links. The objective is to find configurations of the n joint angles where the angles have low variance, giving the arm a smooth appearance. The measures indicate the x-y position of the end of the arm. (3) *Hard maze* [145] considers a robot that navigates a maze for 250 timesteps. We use the Kheperax [98] implementation, where the objective is the robot's energy consumption, and the measures are the final x-y position. The robot is controlled by a neural network with two hidden layers of size 8 and 138 parameters total. In all domains, we linearly transform the objective to the range $[0, 100]$. We consider 100- and 1000-dimensional versions of sphere and arm, yielding five domains: *Sphere 100*, *Sphere 1000*, *Arm 100*, *Arm 1000*, *Maze*.

We select these benchmarks since they are well-studied in the QD literature and exhibit different properties. For instance, Sphere has a separable objective, and its measure space is intentionally distorted to make it difficult to find new archive solutions. In contrast, the variance objective in Arm is non-separable, but its measure space tends to be easier to explore, with prior work [72] showing that even vanilla MAP-Elites fills most of the archive. Finally, as a small-scale QD-RL benchmark, Maze has a less intuitive mapping from neural network parameters to objectives and measures.

Metrics. Our primary metric is *QD score* [190], which holistically measures algorithm performance by summing the objectives of all archive solutions. To ensure no solution subtracts from the score (this happens if objectives are negative), we subtract the minimum objective (i.e., CMA-MAE's min_f) from all solutions' objectives before computing the score. Note that $min_f = 0$ in all domains in this section, but $min_f < 0$ in all environments in Sec. 2.5. We also

record *archive coverage* (fraction of archive cells containing a solution), *best performance* (highest objective in the archive), and *execution time* (wall-clock time of the experiment). In our tables, we abbreviate these metrics as "QD", "Cov", "Best", and "Time."

Procedure. In each domain (Sphere 100, Sphere 1000, Arm 100, Arm 1000, Maze), we conduct a between-groups study with the algorithm (CMA-MAE, LM-MA-MAE, sep-CMA-MAE, OpenAI-MAE) as independent variable and the QD score and execution time as dependent variables. We repeat experiments for 10 trials, where each trial executes an algorithm in a domain for 2 million solution evaluations. All experiments run on a single CPU core, except in Maze, where we run evaluations on an NVIDIA RTX A6000.

Hyperparameters. CMA-MAE and its variants run with archive learning rate $\alpha = 0.001$ (except $\alpha = 0.01$ in Maze) and $\psi = 5$ emitters. Each emitter has batch size $\lambda = 40$ and initial step size $\sigma = 0.02$. LM-MA-MAE sets $k = \lambda = 40$. The Adam optimizer for OpenAI-ES in OpenAI-MAE uses learning rate 0.01 and L2 regularization coefficient 0.005.

Hypotheses. All methods considered model their search distribution with a Gaussian or approximate Gaussian. We predict that methods with a more complex distribution will perform better but take longer to execute. To elaborate, the first and simplest algorithm in this ranking is OpenAI-MAE, which models a fixed isotropic Gaussian. Since this Gaussian has a constant shape that cannot adapt to the search space, we predict OpenAI-MAE will have the lowest performance. However, since OpenAI-MAE only updates the mean of the Gaussian, it should be the fastest algorithm.

Second, sep-CMA-MAE models a diagonal Gaussian. Since this distribution can change shape and adapt over time, we predict it will lead to higher performance when guiding the QD search. While the diagonal Gaussian gives sep-CMA-MAE the same linear complexity as OpenAI-MAE, sep-CMA-MAE will likely be slower, as it requires additional operations to update the diagonal covariance matrix.

Third, LM-MA-MAE uses a rank-k approximation. While the Gaussian in sep-CMA-MAE is limited to being axis-aligned since it is diagonal, the rank-k approximation can represent a more complex Gaussian that is not necessarily axis-aligned. This property should give LM-MA-MAE greater flexibility to adapt to the search space, leading to higher performance. However, the $\Theta(kn)$ complexity will likely make LM-MA-MAE slower than sep-CMA-MAE.

Finally, CMA-MAE maintains a full Gaussian, which should be highly flexible and able to adeptly guide the QD search. The $\Theta(n^2)$ complexity will likely make it the slowest algorithm. Our hypotheses may be summarized as:

H1: *The QD score will be ranked OpenAI-MAE $<$ sep-CMA-MAE $<$ LM-MA-MAE $<$ CMA-MAE.*

H2: *The execution time will be ranked OpenAI-MAE $<$ sep-CMA-MAE $<$ LM-MA-MAE $<$ CMA-MAE.*

2.4.2 Results

Table 2.2 summarizes our results. To analyze the results, we ran an ANOVA for each dependent variable in each domain. Before running the ANOVAs, we verified the data were normally distributed through visual inspection and the Shapiro-Wilk test. Next, we checked homoscedasticity with Levene's test. In homoscedastic settings, we ran a one-way ANOVA, and in non-homoscedastic settings, we ran Welch's one-way ANOVA. In almost all domains, we found significant differences across the algorithms for both dependent variables (Table 2.1). To analyze the rankings in H1 and H2, we performed pairwise comparisons with Tukey's HSD test or a Games-Howell test, depending on whether the data were homoscedastic or not, respectively.

Table 2.1: One-way and Welch's one-way ANOVA results for each dependent variable on each benchmark. All p-values are less than 0.005, except for QD Score in Maze. Note that large between-group variation led to several high F statistics.

	QD Score	Execution Time
Sphere 100	Welch's $F(3, 15.31) = 63\,320$	Welch's $F(3, 18.95) = 57\,608$
Sphere 1000	Welch's $F(3, 15.80) = 688.45$	Welch's $F(3, 15.62) = 3.08 \times 10^6$
Arm 100	$F(3, 36) = 5.46$	Welch's $F(3, 18.48) = 67\,102$
Arm 1000	$F(3, 36) = 258.21$	Welch's $F(3, 19.10) = 2.18 \times 10^6$
Maze	$F(3, 36) = 0.978$ ($p > 0.05$)	$F(3, 36) = 891.73$

H1: In all Sphere and Arm domains, OpenAI-MAE underperformed all other methods. There were no significant differences among the other methods, except that sep-CMA-MAE outperformed CMA-MAE in Sphere 100. In Maze, while there was a trend towards OpenAI-MAE being the best-performing, large variances meant that there were no significant differences among any methods.

Overall, our results fail to support **H1**. Namely, we find that more complex search distributions do not necessarily yield better QD score. On one hand, as predicted, the most basic distribution (OpenAI-MAE's isotropic Gaussian) underperforms CMA-MAE in Sphere and Arm. However, there is no significant difference between OpenAI-MAE and CMA-MAE in Maze. Furthermore, we found no significant differences in any domain between a simple diagonal Gaussian (sep-CMA-MAE) and a full Gaussian (CMA-MAE).

Table 2.2: Optimization benchmark results. We display the metrics described in Sec. 2.4.1 as the mean over 10 trials. QD score is shown as a multiple of 10^6, and Time is measured in minutes.

	Sphere 100				Sphere 1000				Arm 100				Arm 1000				Maze			
	QD	Cov	Best	Time	QD	Cov	Best	Time	QD	Cov	Best	Time	QD	Cov	Best	Time	QD	Cov	Best	Time
CMA-MAE	0.541	0.64	98.82	2.00	0.027	**0.03**	**100.00**	195.60	0.787	**0.79**	**99.98**	2.00	0.763	0.76	99.96	193.41	0.663	0.71	**100.00**	1.06
LM-MA-MAE	0.545	0.65	99.14	1.35	**0.028**	**0.03**	**100.00**	13.42	0.784	**0.79**	**99.98**	1.36	**0.766**	**0.77**	**99.98**	13.29	0.640	0.69	**100.00**	0.72
sep-CMA-MAE	**0.553**	**0.66**	98.74	0.97	**0.028**	**0.03**	**100.00**	3.87	**0.789**	**0.79**	**99.98**	0.98	0.763	0.76	99.96	4.18	0.741	0.79	**100.00**	0.47
OpenAI-MAE	0.007	0.01	**100.00**	**0.66**	0.007	0.01	**100.00**	**2.67**	0.770	0.78	99.97	**0.79**	0.714	0.72	99.96	**3.19**	**0.751**	**0.81**	**100.00**	**0.38**

Table 2.3: Results from training high-dimensional controllers. We display the corrected metrics described in Sec. 2.5.1 as the mean over 10 trials. QD score is shown as a multiple of 10^6, and Time is measured in hours. We also display the memory usage of each algorithm's components (Space), measured in megabytes. Note that Space is constant across all trials.

	QD Ant					QD Half-Cheetah					QD Hopper					QD Walker				
	QD	Cov	Best	Time	Space	QD	Cov	Best	Time	Space	QD	Cov	Best	Time	Space	QD	Cov	Best	Time	Space
LM-MA-MAE	**0.761**	0.36	2,297.91	7.74	112	2.830	0.62	2,243.82	6.45	87	1.135	0.54	2,845.35	4.45	77	0.327	0.42	1,289.64	5.44	84
sep-CMA-MAE	0.730	0.36	2,199.41	8.01	112	**2.892**	**0.63**	2,317.81	**5.77**	87	**1.173**	0.55	**2,884.33**	**3.91**	77	0.325	0.41	1,258.44	4.67	84
OpenAI-MAE	0.638	0.34	2,073.62	**7.25**	111	2.675	0.61	2,172.15	6.30	86	1.153	0.52	2,656.07	4.51	77	0.360	0.45	1,262.87	5.67	84
PGA-ME	0.582	0.34	**2,711.65**	18.19	374	2.682	0.56	**2,722.17**	18.19	325	1.002	0.53	2,740.55	11.89	216	**0.865**	0.50	**2,685.16**	12.42	291
CMA-MEGA (ES)	0.591	0.37	2,122.80	7.43	**110**	2.574	0.59	2,238.03	7.14	**85**	0.502	0.53	1,508.15	3.93	**76**	0.145	0.47	781.89	**3.78**	**83**
CMA-MEGA (TD3, ES)	0.598	**0.40**	2,349.77	14.92	374	2.693	0.59	2,495.93	18.85	325	0.935	0.52	2,498.05	11.18	216	0.789	**0.54**	2,291.46	10.96	291
ME-ES	0.138	0.14	955.12	9.28	111	0.805	0.42	848.60	10.09	86	0.185	0.42	1,043.35	4.46	77	0.037	0.30	388.58	4.32	84
MAP-Elites	0.444	0.38	1,160.22	7.28	**110**	2.371	0.60	1,712.17	7.09	**85**	0.833	**0.56**	2,420.35	4.40	**76**	0.139	0.51	753.96	5.53	**83**

Table 2.4: Results from varying the archive learning rate α in sep-CMA-MAE in the locomotion environments. We display the corrected metrics described in Sec. 2.5.1 as the mean over 10 trials. QD score is shown as a multiple of 10^6.

	QD Ant			QD Half-Cheetah			QD Hopper			QD Walker		
	QD	Cov	Best	QD	Cov	Best	QD	Cov	Best	QD	Cov	Best
$\alpha = 0.0$	0.314	0.13	**3,001.76**	1.577	0.48	2,106.15	0.232	0.31	1,177.58	0.109	0.24	1,019.89
$\alpha = 0.001$	**0.730**	0.36	2,199.41	2.892	0.63	2,317.81	**1.173**	0.55	**2,884.33**	0.325	0.41	1,258.44
$\alpha = 0.01$	0.629	**0.41**	1,914.61	2.839	0.62	2,260.56	1.099	0.56	2,670.70	**0.358**	0.52	**1,318.63**
$\alpha = 0.1$	0.595	0.38	2,285.24	**2.939**	0.63	2,413.25	1.054	**0.57**	2,696.78	0.318	**0.53**	1,277.05
$\alpha = 1.0$	0.387	0.31	2,132.11	2.897	**0.63**	**2,464.52**	0.647	0.55	1,964.28	0.147	0.48	853.18

The only case where increasing search distribution complexity increases performance is with sep-CMA-MAE outperforming OpenAI-MAE in Sphere and Arm. Yet, increasing the complexity further (i.e., LM-MA-MAE's rank-k approximation and CMA-MAE's full Gaussian) fails to garner further improvement.

H2: Pairwise comparisons found that the execution time of the algorithms matched the rankings in **H2**. The difference between CMA-MAE and the variants was particularly pronounced in the higher-dimensional Sphere 1000 and Arm 1000, where, on average, CMA-MAE took 14.5 times longer than LM-MA-MAE, the slowest variant. These results validate **H2**, showing that the variants are empirically faster to run than CMA-MAE, and that the variants become faster as their search distribution becomes simpler.

2.5 Training High-Dimensional Controllers

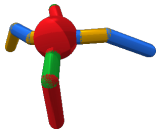
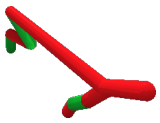
We evaluate our CMA-MAE variants' abilities to train diverse, high-performing neural network controllers for robotic locomotion tasks in the QDGym benchmark [174].

2.5.1 Experimental Setup

Environments. Table 2.5 shows the QDGym environments considered in this work. These environments are *unidirectional*, i.e., the objective is to walk forward quickly, and the measures track the proportion of time that each of the robot's feet touches the ground, e.g., if a robot has four legs, it has four measures. As prior work [239] notes, the challenge in these environments arises from performing objective optimization across the entire archive. Namely, it is easy to find a single high-performing controller and fill the rest of the archive with controllers that stand in place and lift their legs to achieve different measures. However, it is difficult to make the robot walk quickly at all points in the measure space.

As in prior work [239, 175], each domain uses an archive with grid cells. The robot controller is a neural network mapping states to actions. The network has two hidden layers of size 128 and tanh activations and is initialized with Xavier initialization. For the minimum objective min_f, QDGym does not have predefined minimum objectives, but we adopt values from prior work [239] that recorded the minimum objective inserted into an archive during their experiments. Table 2.5 includes the archive dimensions, number of parameters, and minimum objective in each domain.

Table 2.5: QDGym locomotion environments [174].

	QD Ant	QD Half-Cheetah	QD Hopper	QD Walker
				
Archive Grid	[6,6,6,6]	[32,32]	[1024]	[32,32]
Parameters	21,256	20,742	18,947	20,230
Min. Objective	-374.70	-2,797.52	-362.09	-67.17

Baselines. We compare our variants with five baselines: PGA-ME [175], two CMA-MEGA variants [239] that approximate gradients (CMA-MEGA (ES) and CMA-MEGA (TD3, ES)), ME-ES [42], and MAP-Elites. We adopt hyperparameters from the original papers for PGA-ME, the CMA-MEGA variants, and ME-ES, except ME-ES uses a population size of 200. Our MAP-Elites baseline uses isotropic Gaussian noise mutations with standard deviation $\sigma = 0.02$ and batch size 100. The CMA-MAE variants themselves use the same parameters as in the optimization benchmarks (Sec. 2.4.1).

Procedure. We conduct a between-groups study in each environment (QD Ant, QD Half-Cheetah, QD Hopper, QD Walker) with the algorithm (LM-MA-MAE, sep-CMA-MAE, OpenAI-MAE, PGA-ME, CMA-MEGA (ES), CMA-MEGA (TD3, ES), ME-ES, MAP-Elites) as independent variable and QD score and execution time as dependent variables. We repeat experiments for 10 trials, where each trial executes an algorithm for 1 million solution evaluations. Each algorithm runs single-threaded and has 100 CPUs allocated for solution evaluations on a high-performance cluster. In addition to these 100 CPUs, PGA-ME and CMA-MEGA (TD3, ES) are allocated one NVIDIA Tesla P100 GPU to train TD3.

Corrected Metrics. To save computation, we evaluate each solution for only one episode. However, unlike the optimization benchmarks, the locomotion environments are stochastic since each episode's initial state is randomly sampled. Thus, solutions may be inserted into archives due to inaccurate evaluations, e.g., a solution may obtain a high objective by chance. Hence, we report *corrected metrics* [68, 67], where we first re-evaluate all solutions in each final archive for 10 episodes, inserting them into a new, *corrected* archive based on their mean scores. We then compute the metrics from Sec. 2.4.1 over this corrected archive.

Hypotheses. Among the variants, our performance (QD score) prediction remains the same as on the optimization benchmarks (Sec. 2.4.1), i.e., the variant with the simplest search distribution (OpenAI-MAE) will perform worst, and more powerful search distributions (sep-CMA-MAE followed by LM-MA-MAE) will improve performance. While the optimization benchmark results (Sec. 2.4.2) did not support this prediction, higher dimensionality in the locomotion environments may highlight differences between the variants.

Compared to the baselines, we believe the smooth improvement ranking in the variants will enable balancing objective optimization and measure space exploration, yielding better performance. To elaborate, CMA-MEGA (ES) and CMA-MEGA (TD3, ES) use a standard MAP-Elites archive (equivalent to setting $\alpha = 1$ in CMA-MAE's soft archive), so we think they will focus too much on exploration. PGA-ME and ME-ES separate measure space exploration from objective optimization with distinct operations; this separation may be less effective than blending the two aspects.

We predict that execution time differences among the variants will be the same as on the optimization benchmarks; i.e., variants with simpler search distributions will have faster runtimes. Compared to the baselines, we believe a key factor will be whether a method includes deep RL components. Unlike the CMA-MAE variants, PGA-ME and CMA-MEGA (TD3, ES) include components of TD3 [81] to train actor and critic networks, and these training steps are often time-consuming. Our hypotheses may be summarized as follows:

H3: *The performances of the CMA-MAE variants will be ordered as OpenAI-MAE $<$ sep-CMA-MAE $<$ LM-MA-MAE.*

H4: *All CMA-MAE variants will outperform all baselines.*

H5: *The execution times of the CMA-MAE variants will be ordered as OpenAI-MAE $<$ sep-CMA-MAE $<$ LM-MA-MAE.*

H6: *All CMA-MAE variants will be faster than deep RL-based baselines, i.e., PGA-ME and CMA-MEGA (TD3, ES).*

2.5.2 Results

Table 2.3 summarizes our results. Following our analysis procedure in Sec. 2.4.2, we first verified normality through visual inspection and the Shapiro-Wilk test. Next, Levene's test showed homoscedasticity was violated in all environments. Thus, we ran Welch's one-way ANOVA (Table 2.6), finding significant differences in all cases. Finally, we performed pairwise comparisons with the Games-Howell test.

Table 2.6: Welch's one-way ANOVA results in each locomotion environment. All p-values are less than 0.001.

	QD Score	Execution Time
QD Ant	Welch's $F(7, 30.30) = 287.34$	Welch's $F(7, 29.19) = 54.83$
QD Half-Cheetah	Welch's $F(7, 30.66) = 207.25$	Welch's $F(7, 29.55) = 97.66$
QD Hopper	Welch's $F(7, 30.33) = 1017.19$	Welch's $F(7, 30.39) = 60.56$
QD Walker	Welch's $F(7, 29.29) = 500.89$	Welch's $F(7, 29.88) = 111.61$

H3: Table 2.7 shows pairwise comparisons of corrected QD scores for **H3** and **H4**. We find **H3** unsupported, as there tends to be no significant difference among the variants. While these results do not align with Sec. 2.4.2's findings that OpenAI-MAE often underperforms the other variants, both experiments show that more complex search distributions do not necessarily yield higher performance.

H4: Table 2.7 shows that the CMA-MAE variants outperform or are not significantly different from prior ES-based methods (CMA-MEGA (ES) and ME-ES), making them the highest-performing ES-based methods in QD-RL. Compared to deep RL-based methods PGA-ME and CMA-MEGA (TD3, ES), the variants also tend to perform better or have no significant difference. In particular, both sep-CMA-MAE and LM-MA-MAE outperform PGA-ME on QD Ant and QD Hopper while having no significant difference in QD Half-Cheetah. While the variants underperform the deep RL-based methods on QD Walker, prior work [239] highlights the importance of deep RL in this task, as only algorithms with TD3 have performed well here. In short, these results partially support **H4**, showing that the variants often but not always outperform the baselines.

H5: H5 was not supported. We found no significant differences between the variants' runtimes, except sep-CMA-MAE was significantly faster than the other variants in QD Walker. This outcome may arise from the more complex hardware setup of this experiment. Compared to the single CPU used to run the optimization benchmarks, the evaluations here run on 100 CPUs across multiple nodes. Slight differences among the nodes may create runtime variance that obscures differences caused by the search distribution complexity.

Table 2.7: Pairwise comparisons for QD score among the variants (for **H3**) and between the variants and the baselines (for **H4**) in the locomotion environments. Each entry compares the method in the row to the method in the column; e.g., LM-MA-MAE was significantly better than OpenAI-MAE in QD Ant. The symbols used are $<$ (significantly less), $-$ (no significant difference), $>$ (significantly greater), $\varnothing$ (invalid comparison). We abbreviate CMA-MEGA to MEGA for brevity.

	QD Ant								QD Half-Cheetah								QD Hopper								QD Walker							
	LM-MA-MAE	sep-CMA-MAE	OpenAI-MAE	PGA-ME	MEGA (ES)	MEGA (TD3, ES)	ME-ES	MAP-Elites	LM-MA-MAE	sep-CMA-MAE	OpenAI-MAE	PGA-ME	MEGA (ES)	MEGA (TD3, ES)	ME-ES	MAP-Elites	LM-MA-MAE	sep-CMA-MAE	OpenAI-MAE	PGA-ME	MEGA (ES)	MEGA (TD3, ES)	ME-ES	MAP-Elites	LM-MA-MAE	sep-CMA-MAE	OpenAI-MAE	PGA-ME	MEGA (ES)	MEGA (TD3, ES)	ME-ES	MAP-Elites
LM-MA-MAE	$\varnothing$	$-$	$>$	$>$	$>$	$>$	$>$	$>$	$\varnothing$	$-$	$-$	$-$	$-$	$-$	$>$	$>$	$\varnothing$	$-$	$-$	$>$	$>$	$>$	$>$	$>$	$\varnothing$	$-$	$-$	$<$	$>$	$<$	$>$	$>$
sep-CMA-MAE	$-$	$\varnothing$	$-$	$>$	$-$	$-$	$>$	$>$	$-$	$\varnothing$	$-$	$-$	$-$	$-$	$>$	$>$	$-$	$\varnothing$	$-$	$>$	$>$	$>$	$>$	$>$	$-$	$\varnothing$	$-$	$<$	$>$	$<$	$>$	$>$
OpenAI-MAE	$<$	$-$	$\varnothing$	$-$	$-$	$-$	$>$	$>$	$-$	$-$	$\varnothing$	$-$	$-$	$-$	$>$	$>$	$-$	$-$	$\varnothing$	$>$	$>$	$>$	$>$	$>$	$-$	$-$	$\varnothing$	$<$	$>$	$<$	$>$	$>$

H6: All CMA-MAE variants were significantly faster than PGA-ME and CMA-MEGA (TD3, ES) in all domains. The two deep RL-based algorithms took more than twice as long to run as the variants. While variance in compute nodes may have contributed to this difference as we believe it did in **H5**, we believe the majority of the difference stems from the internal algorithm runtime, specifically the aforementioned network training performed in the deep RL-based methods.

Memory Usage: To better understand resource requirements, we report the memory usage of each algorithm's internal components in Table 2.3. Many algorithms have similar usage due to creating similarly sized components. For instance, in the CMA-MAE variants, CMA-MEGA (ES), ME-ES, and MAP-Elites, memory is dominated by the archive, with negligible space for components like emitters. Meanwhile, PGA-ME and CMA-MEGA (TD3, ES) require more memory to store their TD3 replay buffers.

2.5.3 Ablation of Archive Learning Rate

We believe the soft archive and improvement ranking play a key role in the CMA-MAE variants' performance. Thus, we ablate this mechanism by varying the archive learning rate α in sep-CMA-MAE. Table 2.4 shows the result of varying $\alpha \in [0, 1]$; note that all experiments thus far used $\alpha = 0.001$. These results show that, similar to CMA-MAE in benchmark QD domains [72], performance (QD score) falls at the extreme values $\alpha = 0$ and $\alpha = 1$, when sep-CMA-MAE focuses entirely on objective optimization or archive exploration, respectively. In contrast, intermediate values blend both aspects to achieve high performance.

2.6 Discussion and Conclusion

We create variants of CMA-MAE that scale to neural network controllers for robotic locomotion by replacing CMA-MAE's CMA-ES component with efficient approximations. Our results on optimization benchmarks (Sec. 2.4) help distinguish the variants' properties, while our results on locomotion tasks (Sec. 2.5) showcase the effectiveness of the variants compared to existing methods. Furthermore, compared to state-of-the-art deep RL-based methods, our variants bring attractive practical benefits:

(1) The CMA-MAE variants are light on computation. PGA-ME and CMA-MEGA (TD3, ES) both train deep RL components with TD3, a lengthy process that significantly increases runtime as shown in the results of **H6**.

(2) The CMA-MAE variants have very few hyperparameters since they depend on CMA-ES and its variants, which are designed to be parameterized by only an initial step size σ and batch size λ. Hence, the CMA-MAE variants only require 5 hyperparameters ($\psi, \lambda, \sigma, \alpha, min_f$, see Algorithm 1). In contrast, deep RL-based methods require many more parameters: 18 for PGA-ME, 15 for CMA-MEGA (TD3, ES). Methods without deep RL require fewer hyperparameters: 5 for CMA-MEGA (ES), 6 for ME-ES, 2 for MAP-Elites.[†] However, our experiments show that such methods do not perform as well as the CMA-MAE variants.

We emphasize that our CMA-MAE variants are black-box methods that do not leverage the MDP structure of the QD-RL problem, making them suitable for settings beyond QD-RL. Hence, we envision future applications of our variants in areas such as manipulation [164] and scenario generation [19].

[†]These counts are based on prior listings [239] of hyperparameters.

Chapter 3

Adapting Differentiable Quality Diversity to Reinforcement Learning

3.1 Introduction

We focus on the problem of extending differentiable quality diversity (DQD) to reinforcement learning (RL) domains. We propose to approximate gradients for the objective and measure functions, resulting in two variants of the DQD algorithm CMA-MEGA [73].

Consider a half-cheetah agent (Fig. 3.1) trained for locomotion, where the agent must continue walking forward even when one foot is damaged. If we frame this challenge as an RL problem, two approaches to design a robustly capable agent would be to (1) design a reward function and (2) apply domain randomization [241, 184]. However, prior work [120, 39] suggests that designing such a reward function is difficult, while domain randomization may require manually selecting hundreds of environment parameters [184, 176].

As an alternative approach, consider that we have intuition on what behaviors would be useful for adapting to damage. For instance, we can *measure* how often each foot is used during training, and we can pre-train a collection of policies that are diverse in how the agent uses its feet. When one of the agent's feet is damaged during deployment, the agent can adapt to the damage by selecting a policy that did not move the damaged foot during training [49, 42].

Pre-training such a collection of policies may be viewed as a quality diversity (QD) optimization problem [190, 49, 169, 42]. Formally, QD assumes an objective function f and one or more measure functions $\boldsymbol{m}$. The goal of QD is to find solutions satisfying all output combinations of $\boldsymbol{m}$, i.e. moving different combinations of feet, while maximizing each solution's f, i.e. walking forward quickly. Most QD algorithms treat f and $\boldsymbol{m}$ as black boxes, but recent work [73] proposes differentiable quality diversity (DQD), which assumes f and $\boldsymbol{m}$ are differentiable functions with exact

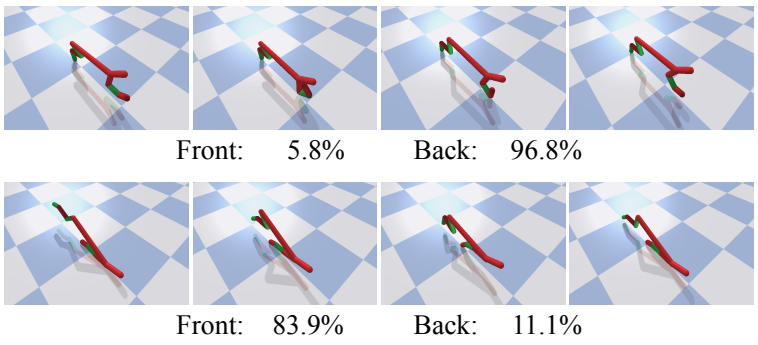


Figure 3.1: A half-cheetah agent executing two walking policies. In the top row, the agent walks on its back foot while tapping the ground with its front foot. In the bottom row, the agent walks on its front foot while jerking its back foot. Values below each row show the percentage of time each foot contacts the ground (each foot is measured individually, so values do not sum to 100%). With these policies, the agent could continue walking even if one foot is damaged.

gradient information. QD algorithms have been applied to procedural content generation [95], robotics [49, 169], aerodynamic shape design [83], and scenario generation in human-robot interaction [71, 74].

The recently proposed DQD algorithm CMA-MEGA [73] outperforms QD algorithms by orders of magnitude when exact gradients are available, such as when searching the latent space of a generative model. However, RL problems like the half-cheetah lack these gradients because the environment is typically non-differentiable, thus limiting the applicability of DQD. To address this limitation, we draw inspiration from how evolution strategies (ES) [4, 258, 205, 157] and deep RL actor-critic methods [214, 216, 149, 81] optimize a reward objective by approximating gradients for gradient descent. *Our key insight is to approximate objective and measure gradients for DQD algorithms by adapting ES and actor-critic methods.*

This chapter makes three contributions. **(1)** We formalize the problem of quality diversity for reinforcement learning (QD-RL) and reduce it to an instance of DQD. **(2)** We develop two QD-RL variants of the DQD algorithm CMA-MEGA, where each algorithm approximates objective and measure gradients with a different combination of ES and actor-critic methods. **(3)** We benchmark our variants on four PyBullet locomotion tasks from QDGym [63, 174]. One variant performs comparably (in terms of QD score; Sec. 3.5.1.3) to the state-of-the-art PGA-ME [175] in two tasks. The other variant achieves comparable QD score with PGA-ME in all tasks* but is less efficient than PGA-ME in two tasks.

*We note that the performance of the CMA-MEGA is worse than PGA-ME in two of the tasks, albeit within variance. We consider it likely that additional runs would result in PGA-ME performing significantly better in these tasks. We leave further evaluation for future work.

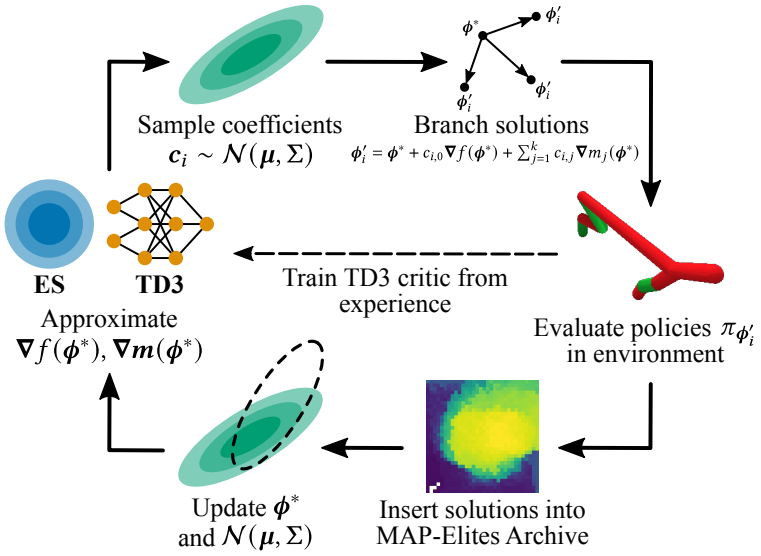


Figure 3.2: We develop two RL variants of the CMA-MEGA algorithm. Similar to CMA-MEGA, the variants sample gradient coefficients $\boldsymbol{c}$ and branch around a solution point ϕ^*. We evaluate each branched solution ϕ'_i as part of a policy $\pi_{\phi'_i}$ and insert ϕ'_i into the archive. We then update ϕ^* and $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ to maximize archive improvement. Our RL variants differ from CMA-MEGA by approximating gradients with ES and TD3, since exact gradients are unavailable in RL settings.

These results contrast with prior work [73] where CMA-MEGA vastly outperforms a DQD algorithm inspired by PGA-ME on benchmark functions where gradient information is available. Overall, we shed light on the limitations of CMA-MEGA in QD domains where the main challenge comes from optimizing the objective rather than from exploring measure space. At the same time, since we decouple gradient estimates from QD optimization, our work opens a path for future research that would benefit from independent improvements to either DQD or RL.

3.2 Problem Statement

3.2.1 Quality Diversity (QD)

We adopt the definition of QD from prior work [73]. For a solution vector $\phi \in \mathbb{R}^n$, QD considers an objective function $f(\phi)$ and k measures[†] $m_i(\phi) \in \mathbb{R}$ (for $i \in 1..k$) or, as a joint measure, $\boldsymbol{m}(\phi) \in \mathbb{R}^k$. These measures form a k-dimensional measure space $\mathcal{X}$. For every $\boldsymbol{x} \in \mathcal{X}$, the QD objective is to find solution ϕ such that $\boldsymbol{m}(\phi) = \boldsymbol{x}$ and $f(\phi)$ is maximized. Since $\mathcal{X}$ is continuous, it would require infinite memory to solve the QD problem, so algorithms in the MAP-Elites family [169, 49] discretize $\mathcal{X}$ by forming a tesselation $\mathcal{Y}$ consisting of M cells. Thus, we relax the QD

[†]Prior work refers to measure function outputs as "behavior characteristics," "behavior descriptors," or "feature descriptors."

problem to one of searching for an *archive* $\mathcal{A}$ consisting of M *elites* $\boldsymbol{\phi}_i$, one for each cell in $\mathcal{Y}$. Then, the QD objective is to maximize the performance $f(\boldsymbol{\phi}_i)$ of all elites:

$$\max_{\boldsymbol{\phi}_{1..M}} \sum_{i=1}^{M} f(\boldsymbol{\phi}_i) \tag{3.1}$$

3.2.1.1 Differentiable Quality Diversity (DQD)

In DQD, we assume f and $\boldsymbol{m}$ are first-order differentiable. We denote the objective gradient as $\boldsymbol{\nabla} f(\boldsymbol{\phi})$, or abbreviated as $\boldsymbol{\nabla} f$, and the measure gradients as $\boldsymbol{\nabla} \boldsymbol{m}(\boldsymbol{\phi})$ or $\boldsymbol{\nabla} \boldsymbol{m}$.

3.2.2 Quality Diversity for Reinforcement Learning (QD-RL)

We define QD-RL as an instance of the QD problem in which each solution $\boldsymbol{\phi}$ parameterizes an RL policy $\pi_{\boldsymbol{\phi}}$. Then, the objective $f(\boldsymbol{\phi})$ is the *expected discounted return* of $\pi_{\boldsymbol{\phi}}$, and the measures $\boldsymbol{m}(\boldsymbol{\phi})$ are functions of $\pi_{\boldsymbol{\phi}}$. Formally, drawing on the Markov Decision Process (MDP) formulation [234], we represent QD-RL as a tuple $(\mathcal{S}, \mathcal{U}, p, r, \gamma, \boldsymbol{m})$. On discrete timesteps t in an episode of interaction, an agent observes state $s \in \mathcal{S}$ and takes action $a \in \mathcal{U}$ according to a policy $\pi_{\boldsymbol{\phi}}(a|s)$. The agent then receives scalar reward $r(s, a)$ and observes next state $s' \in \mathcal{S}$ according to $s' \sim p(\cdot|s, a)$. Each episode thus has a trajectory $\xi = \{s_0, a_0, s_1, a_1, .., s_T\}$, where T is the number of timesteps in the episode, and the probability that policy $\pi_{\boldsymbol{\phi}}$ takes trajectory ξ is $p_{\boldsymbol{\phi}}(\xi) = p(s_0) \prod_{t=0}^{T-1} \pi_{\boldsymbol{\phi}}(a_t|s_t) p(s_{t+1}|s_t, a_t)$. Now, we define the *expected discounted return* of policy $\pi_{\boldsymbol{\phi}}$ as

$$f(\boldsymbol{\phi}) = \mathbb{E}_{\xi \sim p_{\boldsymbol{\phi}}} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right] \tag{3.2}$$

where the discount factor $\gamma \in (0, 1)$ trades off between short- and long-term rewards. Finally, we quantify the behavior of policy $\pi_{\boldsymbol{\phi}}$ via a k-dimensional measure function $\boldsymbol{m}(\boldsymbol{\phi})$.

3.2.2.1 QD-RL as an instance of DQD

We reduce QD-RL to a DQD problem. Since the exact gradients $\boldsymbol{\nabla} f$ and $\boldsymbol{\nabla} \boldsymbol{m}$ usually do not exist in QD-RL, we must instead approximate them.

3.3 Background

3.3.1 Single-Objective Reinforcement Learning

We review algorithms which train a policy to maximize a single objective, i.e. $f(\boldsymbol{\phi})$ in Eq. 3.2, with the goal of applying these algorithms' gradient approximations to DQD in Sec. 3.4.

3.3.1.1 Evolution strategies (ES)

ES [17] is a class of evolutionary algorithms which optimizes the objective by sampling a population of solutions and moving the population towards areas of higher performance. Natural Evolution Strategies (NES) [258, 259] is a type of ES which updates the sampling distribution of solutions by taking steps on distribution parameters in the direction of the natural gradient [6]. For example, with a Gaussian sampling distribution, each iteration of an NES would compute natural gradients to update the mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$.

We consider an NES-inspired algorithm [205] which has demonstrated success in RL domains. This algorithm, which we refer to as OpenAI-ES, samples λ_{es} solutions from an isotropic Gaussian but only computes a gradient step for the mean $\boldsymbol{\phi}$. Each solution sampled by OpenAI-ES is represented as $\boldsymbol{\phi} + \sigma\boldsymbol{\epsilon}_i$, where σ is the fixed standard deviation of the Gaussian and $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Once these solutions are evaluated, OpenAI-ES estimates the gradient as

$$\boldsymbol{\nabla} f(\boldsymbol{\phi}) \approx \frac{1}{\lambda_{es}\sigma} \sum_{i=1}^{\lambda_{es}} f(\boldsymbol{\phi} + \sigma\boldsymbol{\epsilon}_i)\boldsymbol{\epsilon}_i \tag{3.3}$$

OpenAI-ES then passes this estimate to an Adam optimizer [136] which outputs a gradient ascent step for $\boldsymbol{\phi}$. To make the estimate more accurate, OpenAI-ES further includes techniques such as mirror sampling and rank normalization [27, 102, 258].

3.3.1.2 Actor-critic methods

While ES treats the objective as a black box, actor-critic methods leverage the MDP structure of the objective, i.e. the fact that $f(\boldsymbol{\phi})$ is a sum of Markovian values. We are most interested in Twin Delayed Deep Deterministic policy gradient (TD3) [81], an off-policy actor-critic method. TD3 maintains (1) an actor consisting of the policy $\pi_{\boldsymbol{\phi}}$ and (2) a critic consisting of state-action value functions $Q_{\boldsymbol{\theta}_1}(s, a)$ and $Q_{\boldsymbol{\theta}_2}(s, a)$ which differ only in random initialization.

Through interactions in the environment, the actor generates experience which is stored in a replay buffer $\mathcal{B}$. This experience is sampled to train $Q_{\boldsymbol{\theta}_1}$ and $Q_{\boldsymbol{\theta}_2}$. Simultaneously, the actor improves by maximizing $Q_{\boldsymbol{\theta}_1}$ via gradient ascent ($Q_{\boldsymbol{\theta}_2}$ is only used during critic training). Specifically, for an objective f' which is based on the critic and approximates f, TD3 estimates a gradient $\boldsymbol{\nabla} f'(\boldsymbol{\phi})$ and passes it to an Adam optimizer. Notably, TD3 never updates network weights directly, instead accumulating weights into *target networks* $\pi_{\boldsymbol{\phi}'}$, $Q_{\boldsymbol{\theta}'_1}$, $Q_{\boldsymbol{\theta}'_2}$ via an exponentially weighted moving average with update rate τ.

3.3.2 Quality Diversity Algorithms

3.3.2.1 MAP-Elites extensions for QD-RL

One of the simplest QD algorithms is MAP-Elites [169, 49]. MAP-Elites creates an archive $\mathcal{A}$ by tesselating the measure space $\mathcal{X}$ into a grid of evenly-sized cells. Then, it draws λ initial solutions from a multivariate Gaussian $\mathcal{N}(\boldsymbol{\phi_0}, \sigma \boldsymbol{I})$ centered at some $\boldsymbol{\phi}_0$. Next, for each sampled solution $\boldsymbol{\phi}$, MAP-Elites computes $f(\boldsymbol{\phi})$ and $\boldsymbol{m}(\boldsymbol{\phi})$ and inserts $\boldsymbol{\phi}$ into $\mathcal{A}$. In subsequent iterations, MAP-Elites randomly selects λ solutions from $\mathcal{A}$ and adds Gaussian noise, i.e. solution $\boldsymbol{\phi}$ becomes $\boldsymbol{\phi} + \mathcal{N}(\mathbf{0}, \sigma \boldsymbol{I})$. Solutions are placed into cells based on their measures; if a solution has higher f than the solution currently in the cell, it replaces that solution. Once inserted into $\mathcal{A}$, solutions are known as *elites*.

Due to the high dimensionality of neural network parameters, direct policy optimization with MAP-Elites has not proven effective in QD-RL [42], although indirect encodings have been shown to scale to large policy networks [195, 84]. For direct search, several extensions merge MAP-Elites with actor-critic methods and ES. For instance, Policy Gradient Assisted MAP-Elites (PGA-ME) [175] combines MAP-Elites with TD3. Each iteration, PGA-ME evaluates λ solutions for insertion into the archive. $\frac{\lambda}{2}$ of these are created by selecting random solutions from the archive and taking gradient ascent steps with a TD3 critic. The other $\frac{\lambda}{2}$ solutions are created with a directional variation operator [249] which selects two solutions $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$ from the archive and creates a new one according to $\boldsymbol{\phi}' = \boldsymbol{\phi}_1 + \sigma_1 \mathcal{N}(\mathbf{0}, \boldsymbol{I}) + \sigma_2 (\boldsymbol{\phi}_2 - \boldsymbol{\phi}_1) \mathcal{N}(0, 1)$. Finally, PGA-ME maintains a "greedy actor" which provides actions when training the critics (identically to the actor in TD3). Every iteration, PGA-ME inserts this greedy actor into the archive. PGA-ME achieves state-of-the-art performance on locomotion tasks in the QDGym benchmark [174].

Another MAP-Elites extension is ME-ES [42], which combines MAP-Elites with an OpenAI-ES optimizer. In the "explore-exploit" variant, ME-ES alternates between two phases. In the "exploit" phase, ME-ES restarts OpenAI-ES

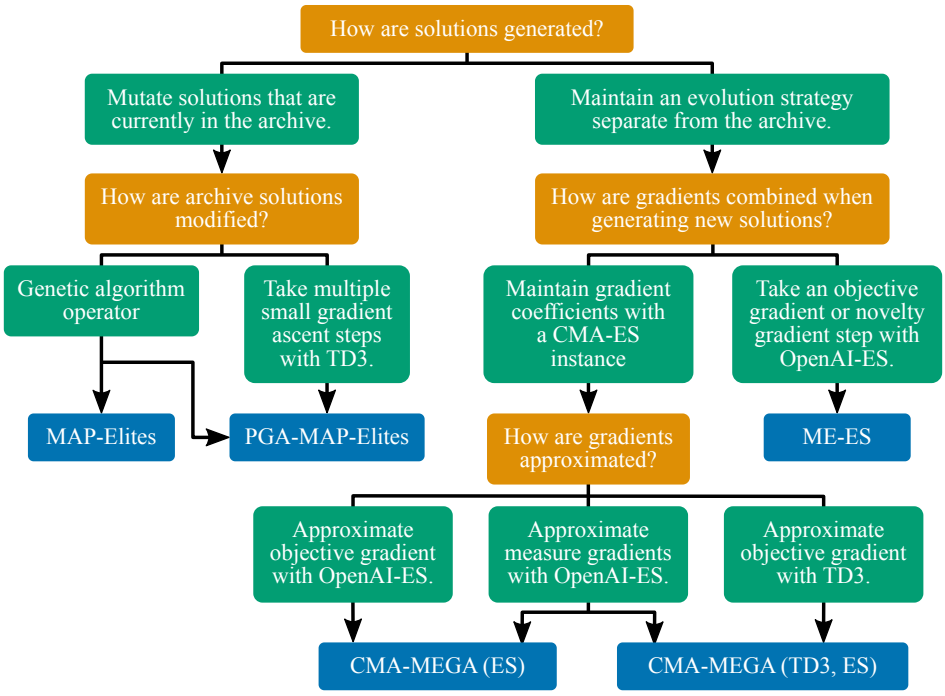


Figure 3.3: Diagram of MAP-Elites extensions for QD-RL, showing how our CMA-MEGA variants differ from other QD-RL algorithms.

at a mean ϕ and optimizes the objective for k iterations, inserting the current ϕ into the archive in each iteration. In the "explore" phase, ME-ES repeats this process, but OpenAI-ES instead optimizes for novelty, where novelty is the distance in measure space from a new solution to previously encountered solutions. ME-ES also has an "exploit" variant and an "explore" variant, which each execute only one type of phase.

Our work is related to ME-ES in that we also adapt OpenAI-ES, but instead of alternating between following a novelty gradient and objective gradient, we compute all objective and measure gradients and allow a CMA-ES [108] instance to decide which gradients to follow by sampling gradient coefficients from a multivariate Gaussian updated over time (Sec. 3.3.2.2). We include MAP-Elites, PGA-ME, and ME-ES as baselines in our experiments. Refer to Fig. 3.3 for a diagram which compares these algorithms to our approach.

3.3.2.2 Covariance Matrix Adaptation MAP-Elites via a Gradient Arborescence (CMA-MEGA)

We directly extend CMA-MEGA [73] to address QD-RL. CMA-MEGA is a DQD algorithm based on the QD algorithm CMA-ME [77]. The intuition behind CMA-MEGA is that if we knew which direction the current solution point ϕ^* should move in objective-measure space, then we could calculate that change in search space via a linear combination

of objective and measure gradients. From CMA-ME, we know a good direction is one that results in the largest archive improvement.

Each iteration, CMA-MEGA first calculates objective and measure gradients for a solution point $\boldsymbol{\phi}^*$. Next, it generates λ new solutions by sampling gradient coefficients $\boldsymbol{c} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and computing $\boldsymbol{\phi}' \leftarrow \boldsymbol{\phi}^* + c_0 \boldsymbol{\nabla} f(\boldsymbol{\phi}^*) + \sum_{j=1}^{k} c_j \boldsymbol{\nabla} m_j(\boldsymbol{\phi}^*)$. CMA-MEGA inserts these solutions into the archive and computes their *improvement,* Δ. Δ is defined as $f(\boldsymbol{\phi}')$ if $\boldsymbol{\phi}'$ populates a new cell, and $f(\boldsymbol{\phi}') - f(\boldsymbol{\phi}'_{\mathcal{E}})$ if $\boldsymbol{\phi}'$ improves an existing cell (replaces a previous solution $\boldsymbol{\phi}'_{\mathcal{E}}$). After CMA-MEGA inserts the solutions, it ranks them by Δ. If a solution populates a new cell, its Δ always ranks higher than that of a solution which only improves an existing cell. CMA-MEGA then moves the solution point $\boldsymbol{\phi}^*$ towards the largest archive improvement, but also adapts the distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ towards better gradient coefficients by the same ranking. By leveraging gradient information, CMA-MEGA solves QD benchmarks with orders of magnitude fewer solution evaluations than previous QD algorithms.

3.3.2.3 Beyond MAP-Elites

Several QD-RL algorithms have been developed outside the MAP-Elites family. NS-ES [44] builds on Novelty Search (NS) [145, 146], a family of QD algorithms which add solutions to an *unstructured archive* only if they are far away from existing archive solutions in measure space. Using OpenAI-ES, NS-ES concurrently optimizes several agents for novelty. Its variants NSR-ES and NSRA-ES optimize for a linear combination of novelty and objective. Meanwhile, QD-PG [186] maintains an archive with all past solutions and optimizes agents along a Pareto front of the objective and novelty. Finally, Diversity via Determinants (DvD) [182] leverages a kernel method to maintain diversity in a population of solutions. As NS-ES, QD-RL, and DvD do not output a MAP-Elites grid archive, we leave their investigation for future work.

3.3.3 Diversity in Reinforcement Learning

Here we distinguish QD-RL from prior work which also applies diversity to RL. One area of work is in latent- and goal-conditioned policies. For latent-conditioned policy $\pi_{\boldsymbol{\phi}}(a|s, z)$ [65, 139, 147] or goal-conditioned policy $\pi_{\boldsymbol{\phi}}(a|s, g)$ [210, 9], varying the latent variable z or goal g results in different behaviors, e.g. different walking gaits or walking to a different location. While QD-RL also seeks a range of behaviors, the measures $\boldsymbol{m}(\boldsymbol{\phi})$ are computed *after* evaluating $\boldsymbol{\phi}$,

rather than *before* the evaluation. In general, QD-RL focuses on finding a variety of policies for a single task, rather than attempting to solve a variety of tasks with a single conditioned policy.

Another area of work combines evolutionary and actor-critic algorithms to solve single-objective hard-exploration problems [43, 132, 188, 236, 131]. In these methods, an evolutionary algorithm such as cross-entropy method [20] facilitates exploration by generating a diverse population of policies, while an actor-critic algorithm such as TD3 trains high-performing policies with this population's environment experience. QD-RL differs from these methods in that it views diversity as a component of the output, while these methods view diversity as a means for environment exploration. Hence, QD-RL measures policy behavior via a measure function and collects diverse policies in an archive. In contrast, these RL exploration methods assume that trajectory diversity, rather than targeting specific behavioral diversity, is enough to drive exploration to discover a single optimal policy.

3.4 Approximating Gradients for DQD

Since DQD algorithms require exact objective and measure gradients, we cannot directly apply CMA-MEGA to QD-RL. To address this limitation, we replace exact gradients with gradient approximations (Sec. 3.4.1) and develop two CMA-MEGA variants (Sec. 3.4.2).

3.4.1 Approximating Objective and Measure Gradients

We adapt gradient approximations from ES and actor-critic methods. Since the objective has an MDP structure, we estimate objective gradients $\boldsymbol{\nabla} f$ with ES and actor-critic methods. Since the measures are black boxes, we estimate measure gradients $\boldsymbol{\nabla} \boldsymbol{m}$ with ES.

3.4.1.1 Approximating objective gradients with ES and actor-critic methods

We estimate objective gradients with two methods. First, we treat the objective as a black box and estimate its gradient with a black box method, namely the OpenAI-ES gradient estimate in Eq. 3.3. Since OpenAI-ES performs well in RL domains [205, 178, 143], we believe this estimate is suitable for approximating gradients for CMA-MEGA in QD-RL settings. Importantly, this estimate requires environment interaction by evaluating λ_{es} solutions.

Since the objective has a well-defined structure, i.e. it is a sum of rewards from an MDP (Eq. 3.2), we also estimate its gradient with an actor-critic method, TD3. TD3 is well-suited for this purpose because it efficiently estimates objective gradients for the multiple policies that CMA-MEGA and other QD-RL algorithms generate. In particular, once the critic is trained, TD3 can provide a gradient estimate for any policy without additional environment interaction.

Among actor-critic methods, we select TD3 since it achieves high performance while optimizing primarily for the RL objective. Prior work [81] shows that TD3 outperforms on-policy actor-critic methods [214, 216]. While the off-policy Soft Actor-Critic [103] algorithm can outperform TD3, it optimizes a maximum-entropy objective designed to encourage exploration. In our work, we explore by finding policies with different measures. Thus, we leave for future work the problem of integrating QD-RL with the action diversity encouraged by entropy maximization.

3.4.1.2 Approximating measure gradients with ES

Since measures do not have special properties such as an MDP structure (Sec. 3.2.2), we only estimate their gradient with black box methods. Thus, similar to the objective, we approximate each measure's gradient $\boldsymbol{\nabla} m_i$ with the OpenAI-ES gradient estimate, replacing f with m_i in Eq. 3.3.

Since the OpenAI-ES gradient estimate requires additional environment interaction, all of our CMA-MEGA variants require environment interaction to estimate gradients. However, the environment interaction required to estimate measure gradients remains constant even as the number of measures increases, since we can reuse the same λ_{es} solutions to estimate each $\boldsymbol{\nabla} m_i$.

In problems where the measures have an MDP structure similar to the objective, it may be feasible to estimate each $\boldsymbol{\nabla} m_i$ with its own TD3 instance. In the environments in our work (Sec. 3.5.1), each measure is non-Markovian since it calculates the proportion of time a walking agent's foot spends on the ground. This calculation depends on the entire agent trajectory rather than on one state.

3.4.2 CMA-MEGA Variants

Our choice of gradient approximations leads to two CMA-MEGA variants. **CMA-MEGA (ES)** approximates objective and measure gradients with OpenAI-ES, while **CMA-MEGA (TD3, ES)** approximates the objective gradient with TD3 and measure gradients with OpenAI-ES. Fig. 3.2 shows an overview of both algorithms, and Algorithm 2 shows their

Algorithm 2: CMA-MEGA (ES) and CMA-MEGA (TD3, ES). Highlighted portions are only executed in CMA-MEGA (TD3, ES). Adapted from CMA-MEGA [73]. Functions whose names are in SMALL_CAPS are helper functions and are available in Algorithm 3, Algorithm 4, and Algorithm 5.

1 **CMA-MEGA variants** $(evaluate, \boldsymbol{\phi}_0, N, \lambda, \sigma_g, \eta, \lambda_{es}, \sigma_e)$**:**
  Input: Function $evaluate$ which executes a policy $\boldsymbol{\phi}$ and outputs objective $f(\boldsymbol{\phi})$ and measures $\boldsymbol{m}(\boldsymbol{\phi})$, initial solution $\boldsymbol{\phi}_0$, desired iterations N, batch size λ, initial CMA-ES step size σ_g, learning rate η, ES batch size λ_{es}, ES standard deviation σ_e
  Result: Generates $N\lambda$ solutions, storing elites in an archive $\mathcal{A}$
2 $\lambda' \leftarrow \lambda - 1 - 1$
3 Initialize empty archive $\mathcal{A}$, solution point $\boldsymbol{\phi}^* \leftarrow \boldsymbol{\phi}_0$
4 Initialize CMA-ES with population λ', resulting in $\boldsymbol{\mu} = \mathbf{0}, \boldsymbol{\Sigma} = \sigma_g \boldsymbol{I}$, and internal CMA-ES parameters $\boldsymbol{p}$
5 $\mathcal{B}, Q_{\boldsymbol{\theta}_1}, Q_{\boldsymbol{\theta}_2}, \pi_{\boldsymbol{\phi}_q}, Q_{\boldsymbol{\theta}'_1}, Q_{\boldsymbol{\theta}'_2}, \pi_{\boldsymbol{\phi}'_q} \leftarrow$ INITIALIZE_TD3()
6 **for** $iter \leftarrow 1..N$ **do**
7 $f(\boldsymbol{\phi}^*), \boldsymbol{m}(\boldsymbol{\phi}^*) \leftarrow evaluate(\boldsymbol{\phi}^*)$
8 UPDATE_ARCHIVE$(\mathcal{A}, \boldsymbol{\phi}^*, f(\boldsymbol{\phi}^*), \boldsymbol{m}(\boldsymbol{\phi}^*))$
9 $\nabla f(\boldsymbol{\phi}^*), \nabla \boldsymbol{m}(\boldsymbol{\phi}^*) \leftarrow$ ES_GRADIENTS$(\boldsymbol{\phi}^*, \lambda_{es}, \sigma_e)$
10 $\nabla f(\boldsymbol{\phi}^*) \leftarrow$ TD3_GRADIENT$(\boldsymbol{\phi}^*, Q_{\boldsymbol{\theta}_1}, \mathcal{B})$
11 Normalize $\nabla f(\boldsymbol{\phi}^*)$ and $\nabla \boldsymbol{m}(\boldsymbol{\phi}^*)$ to be unit vectors
12 **for** $i \leftarrow 1..\lambda'$ **do**
13 $\boldsymbol{c}_i \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
14 $\nabla_i \leftarrow c_{i,0} \nabla f(\boldsymbol{\phi}^*) + \sum_{j=1}^{k} c_{i,j} \nabla m_j(\boldsymbol{\phi}^*)$
15 $\boldsymbol{\phi}'_i \leftarrow \boldsymbol{\phi}^* + \nabla_i$
16 $f(\boldsymbol{\phi}'_i), \boldsymbol{m}'(\boldsymbol{\phi}'_i) \leftarrow evaluate(\boldsymbol{\phi}'_i)$
17 $\Delta_i \leftarrow$ UPDATE_ARCHIVE$(\mathcal{A}, \boldsymbol{\phi}'_i, f(\boldsymbol{\phi}'_i), \boldsymbol{m}(\boldsymbol{\phi}'_i))$
18 Rank $\boldsymbol{c}_i, \nabla_i$ by Δ_i
19 Adapt CMA-ES parameters $\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{p}$ based on rankings of $\boldsymbol{c}_i$
20 $\boldsymbol{\phi}^* \leftarrow \boldsymbol{\phi}^* + \eta \sum_{i=1}^{\lambda} w_i \nabla_{\text{rank[i]}}$ `// ` w_i `is part of` $\boldsymbol{p}$
21 **if** there is no change in $\mathcal{A}$ **then**
22 Restart CMA-ES with $\boldsymbol{\mu} = \mathbf{0}, \boldsymbol{\Sigma} = \sigma_g \boldsymbol{I}$
23 Set $\boldsymbol{\phi}^*$ to a randomly selected elite from $\mathcal{A}$
24 $f(\boldsymbol{\phi}_q), \boldsymbol{m}(\boldsymbol{\phi}_q) \leftarrow evaluate(\boldsymbol{\phi}_q)$
25 UPDATE_ARCHIVE$(\mathcal{A}, \boldsymbol{\phi}_q, f(\boldsymbol{\phi}_q), \boldsymbol{m}(\boldsymbol{\phi}_q))$
26 Add experience from all calls to $evaluate$ into $\mathcal{B}$
27 TRAIN_TD3$(Q_{\boldsymbol{\theta}_1}, Q_{\boldsymbol{\theta}_2}, \pi_{\boldsymbol{\phi}_q}, Q_{\boldsymbol{\theta}'_1}, Q_{\boldsymbol{\theta}'_2}, \pi_{\boldsymbol{\phi}'_q}, \mathcal{B})$

pseudocode. As CMA-MEGA (TD3, ES) builds on CMA-MEGA (ES), we present only CMA-MEGA (TD3, ES) and highlight lines that CMA-MEGA (TD3, ES) additionally executes.

Identically to CMA-MEGA, the two variants maintain three primary components: a solution point $\boldsymbol{\phi}^*$, a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ for sampling gradient coefficients, and a MAP-Elites archive $\mathcal{A}$ for storing solutions. We initialize the archive and solution point on line 3, and we initialize the coefficient distribution as part of a CMA-ES instance on line 4.[‡]

[‡]We set the CMA-ES batch size λ' slightly lower than the total batch size λ (line 2). While CMA-MEGA (ES) and CMA-MEGA (TD3, ES) both evaluate λ solutions each iteration, one evaluation is reserved for $\boldsymbol{\phi}^*$ (line 7). In CMA-MEGA (TD3, ES), one more evaluation is reserved for the greedy actor (line 24).

Algorithm 3: Helper function for updating the archive.

UPDATE_ARCHIVE $(\mathcal{A}, \boldsymbol{\phi}, f(\boldsymbol{\phi}), \boldsymbol{m}(\boldsymbol{\phi}))$**:**
  // $\mathcal{E}$ contains $\boldsymbol{\phi}_\mathcal{E}, f(\boldsymbol{\phi}_\mathcal{E}), \boldsymbol{m}(\boldsymbol{\phi}_\mathcal{E})$
  $\mathcal{E} \leftarrow$ cell in $\mathcal{A}$ corresponding to $\boldsymbol{m}$
  if $\mathcal{E}$ is empty **then**
    $\boldsymbol{\phi}_\mathcal{E}, f(\boldsymbol{\phi}_\mathcal{E}), \boldsymbol{m}(\boldsymbol{\phi}_\mathcal{E}) \leftarrow \boldsymbol{\phi}, f(\boldsymbol{\phi}), \boldsymbol{m}(\boldsymbol{\phi})$
    return (NEW_CELL, $f(\boldsymbol{\phi})$)
  else if $f(\boldsymbol{\phi}) > f(\boldsymbol{\phi}_\mathcal{E})$ **then**
    $\boldsymbol{\phi}_\mathcal{E}, f(\boldsymbol{\phi}_\mathcal{E}), \boldsymbol{m}(\boldsymbol{\phi}_\mathcal{E}) \leftarrow \boldsymbol{\phi}, f(\boldsymbol{\phi}), \boldsymbol{m}(\boldsymbol{\phi})$
    return (IMPROVE_EXISTING_CELL, $f(\boldsymbol{\phi}) - f(\boldsymbol{\phi}_\mathcal{E})$)
  else
    return (NOT_ADDED, $f(\boldsymbol{\phi}) - f(\boldsymbol{\phi}_\mathcal{E})$)

Algorithm 4: TD3 helper functions. Adapted from PGA-ME [175] and TD3 [81].

INITIALIZE_TD3:
  $\mathcal{B} \leftarrow$ initialize_replay_buffer()
  // As done in the TD3 author implementation [81], we initialize these networks with the default PyTorch weights.
  $Q_{\boldsymbol{\theta}_1}, Q_{\boldsymbol{\theta}_2}, \pi_{\boldsymbol{\phi}_q} \leftarrow$ initialize_networks()
  $Q_{\boldsymbol{\theta}'_1}, Q_{\boldsymbol{\theta}'_2}, \pi_{\boldsymbol{\phi}'_q} \leftarrow Q_{\boldsymbol{\theta}_1}, Q_{\boldsymbol{\theta}_2}, \pi_{\boldsymbol{\phi}_q}$
  return $\mathcal{B}, Q_{\boldsymbol{\theta}_1}, Q_{\boldsymbol{\theta}_2}, \pi_{\boldsymbol{\phi}_q}, Q_{\boldsymbol{\theta}'_1}, Q_{\boldsymbol{\theta}'_2}, \pi_{\boldsymbol{\phi}'_q}$
TD3_GRADIENT $(\boldsymbol{\phi}, Q_{\boldsymbol{\theta}_1}, \mathcal{B})$**:**
  Sample n_{pg} transitions $(s_t, a_t, r(s_t, a_t), s_{t+1})$ from $\mathcal{B}$
  $\nabla_{\boldsymbol{\phi}} J(\boldsymbol{\phi}) = \frac{1}{n_{pg}} \sum \nabla_{\boldsymbol{\phi}} \pi_{\boldsymbol{\phi}}(s_t) \nabla_a Q_{\boldsymbol{\theta}_1}(s_t, a)|_{a=\pi_{\boldsymbol{\phi}}(s_t)}$
  return $\nabla_{\boldsymbol{\phi}} J(\boldsymbol{\phi})$
TRAIN_TD3 $(Q_{\boldsymbol{\theta}_1}, Q_{\boldsymbol{\theta}_2}, \pi_{\boldsymbol{\phi}_q} Q_{\boldsymbol{\theta}'_1}, Q_{\boldsymbol{\theta}'_2}, \pi_{\boldsymbol{\phi}'_q}, \mathcal{B})$**:**
  // Trains the critic and the greedy actor.
  for $i \leftarrow 1..n_{crit}$ **do**
    Sample n_q transitions $(s_t, a_t, r(s_t, a_t), s_{t+1})$ from $\mathcal{B}$
    // Sample smoothing noise.
    $\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma_p), -c_{clip}, c_{clip})$
    $y = r(s_t, a_t) + \gamma \min_{i=1,2} Q_{\boldsymbol{\theta}'_i}(s_{t+1}, \pi_{\boldsymbol{\phi}'_q}(s_{t+1}) + \epsilon)$
    // Update critics.
    $\boldsymbol{\theta}_i \leftarrow \arg\min_{\boldsymbol{\theta}_i} \frac{1}{n_q} \sum (y - Q_{\boldsymbol{\theta}_i}(s_t, a_t))^2$
    if $t \mod d = 0$ **then**
      // Update greedy actor.
      $\nabla_{\boldsymbol{\phi}_q} J(\boldsymbol{\phi}_q) = \frac{1}{n_q} \sum \nabla_{\boldsymbol{\phi}_q} \pi_{\boldsymbol{\phi}_q}(s_t) \nabla_a Q_{\boldsymbol{\theta}_1}(s_t, a)|_{a=\pi_{\boldsymbol{\phi}_q}(s_t)}$
      // Update targets.
      $\boldsymbol{\theta}'_i \leftarrow \tau \boldsymbol{\theta}_i + (1 - \tau) \boldsymbol{\theta}'_i$
      $\boldsymbol{\phi}'_q \leftarrow \tau \boldsymbol{\phi}_q + (1 - \tau) \boldsymbol{\phi}'_q$

Algorithm 5: Helper function for estimating objective and measure gradients with the gradient estimate from OpenAI-ES. This implementation differs from Eq. 3.3 since it includes mirror sampling and rank normalization.

ES_GRADIENTS $(\boldsymbol{\phi}, \lambda_{es}, \sigma_e)$**:**
  `// Mirror sampling - divide` λ_{es} `by 2.`
  for $i \leftarrow 1..\frac{\lambda_{es}}{2}$ **do**
    $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
    $\boldsymbol{x}_i \leftarrow \boldsymbol{\phi} + \sigma_e \boldsymbol{\epsilon}_i$
    $f(\boldsymbol{x}_i), \boldsymbol{m}(\boldsymbol{x}_i) \leftarrow evaluate(\boldsymbol{x}_i)$
    $\boldsymbol{x}'_i \leftarrow \boldsymbol{\phi} - \sigma_e \boldsymbol{\epsilon}_i$ `//` $\boldsymbol{x}'_i$ `reflects` $\boldsymbol{x}_i$`.`
    $f(\boldsymbol{x}'_i), \boldsymbol{m}(\boldsymbol{x}'_i) \leftarrow evaluate(\boldsymbol{x}_i)$
  for $j \leftarrow 0..k$ **do**
    if $j = 0$ **then**
      $L \leftarrow$ all $\boldsymbol{x}_i$ and $\boldsymbol{x}'_i$, sorted by f
    else
      $L \leftarrow$ all $\boldsymbol{x}_i$ and $\boldsymbol{x}'_i$, sorted by m_j
    `// Rank normalization.`
    $R \leftarrow$ rank (index) of every $\boldsymbol{x}_i$ in L
    $R' \leftarrow$ rank (index) of every $\boldsymbol{x}'_i$ in L
    `// Ranks should be normalized over both lists combined (`$R \| R'$`) rather than in each list separately.`
    Normalize ranks in $R \| R'$ to $[-0.5, 0.5]$
    `// Estimate gradient.`
    $\boldsymbol{\nabla} \leftarrow \frac{1}{\frac{\lambda_{es}}{2}\sigma_e} \sum_{i=1}^{\frac{\lambda_{es}}{2}} \boldsymbol{\epsilon}_i (R_i - R'_i)$
    if $j = 0$ **then**
      $\boldsymbol{\nabla} f(\boldsymbol{\phi}) \leftarrow \boldsymbol{\nabla}$
    else
      $\boldsymbol{\nabla} m_j(\boldsymbol{\phi}) \leftarrow \boldsymbol{\nabla}$
  return $\boldsymbol{\nabla} f(\boldsymbol{\phi}), \boldsymbol{\nabla} \boldsymbol{m}(\boldsymbol{\phi})$

In the main loop (line 6), we follow the workflow shown in Fig. 3.2. First, after evaluating ϕ^* and inserting it into the archive (line 7-8), we approximate its gradients with either ES or TD3 (line 9-10). *This gradient approximation forms the key difference between our variants and the original CMA-MEGA algorithm [73].*

Next, we branch from ϕ^* to create solutions ϕ'_i by sampling $\boldsymbol{c}_i$ from the coefficient distribution and computing perturbations $\boldsymbol{\nabla}_i$ (line 13-15). We then evaluate each ϕ'_i and insert it into the archive (line 16-17).

Finally, we update the solution point and the coefficient distribution's CMA-ES instance by forming an *improvement ranking* based on the improvement Δ_i (Sec. 3.3.2.2; line 18-20). Importantly, since we rank based on improvement, this update enables the CMA-MEGA variants to maximize the QD objective (Eq. 3.1) [73].

Our CMA-MEGA variants have two additional components. First, we check if no solutions were inserted into the archive at the end of the iteration, which would indicate that we should reset the coefficient distribution and the solution point (line 21-23). Second, in the case of CMA-MEGA (TD3, ES), we manage a TD3 instance similar to how PGA-ME does (Sec. 3.3.2.1). This TD3 instance consists of a replay buffer $\mathcal{B}$, critic networks $Q_{\boldsymbol{\theta}_1}$ and $Q_{\boldsymbol{\theta}_2}$, a greedy actor π_{ϕ_q}, and target networks $Q_{\boldsymbol{\theta}'_1}$, $Q_{\boldsymbol{\theta}'_2}$, $\pi_{\phi'_q}$ (all initialized on line 5). At the end of each iteration, we use the greedy actor to train the critics, and we also insert it into the archive (line 24-27).

3.5 Experiments

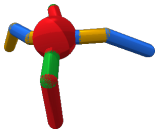
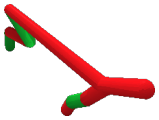
We compare our two proposed CMA-MEGA variants (CMA-MEGA (ES), CMA-MEGA (TD3, ES)) with three baselines (PGA-ME, ME-ES, MAP-Elites) in four locomotion tasks. We implement MAP-Elites as described in Sec. 3.3.2.1, and we select the explore-exploit variant for ME-ES since it has performed at least as well as both the explore variant and the exploit variant in several domains [42].

3.5.1 Evaluation Domains

3.5.1.1 QDGym

We evaluate our algorithms in four locomotion environments from QDGym [174], a library built on PyBullet Gym [45, 63] and OpenAI Gym [28]. Table 2.5 lists all environment details. In each environment, the QD algorithm outputs an archive of walking policies for a simulated agent. The agent is primarily rewarded for its forward speed. There are

Table 3.1: QDGym environments details. We list the dimensions of the state space ($|\mathcal{S}|$) and action space ($|\mathcal{U}|$), number of neural network parameters, number of measures $|\mathcal{X}|$, archive grid dimensions (number of cells along each dimension), total archive grid cells, and min and max objectives (Sec. 3.5.1.3).

	QD Ant	QD Half-Cheetah	QD Hopper	QD Walker
				
$\|\mathcal{S}\|$	28	26	15	22
$\|\mathcal{U}\|$	8	6	3	6
Parameters	21,256	20,742	18,947	20,230
$\|\mathcal{X}\|$	4	2	1	2
Archive dim	[6,6,6,6]	[32,32]	[1024]	[32,32]
Grid cells	1,296	1,024	1,024	1,024
Min objective	-374.70	-2,797.52	-362.09	-67.17
Max objective	2,500.00	3,000.00	2,500.00	2,500.00

also reward shaping [173] signals, such as a punishment for applying higher joint torques, intended to guide policy optimization. The measures compute the proportion of time (number of timesteps divided by total timesteps in an episode) that each of the agent's feet contacts the ground.

QDGym is challenging because the objective in each environment does not "align" with the measures, in that finding policies with different measures (i.e. exploring the archive) does not necessarily lead to optimization of the objective. While it may be trivial to fill the archive with low-performing policies which stand in place and lift the feet up and down to achieve different measures, the agents' complexity (high degrees of freedom) makes it difficult to learn a high-performing policy for each value of the measures.

3.5.1.2 Hyperparameters

Each agent's policy is a neural network which takes in states and outputs actions. There are two hidden layers of 128 nodes, and the hidden and output layers have `tanh` activation. We initialize weights with Xavier initialization [91].

For the archive, we tessellate each environment's measure space into a grid of evenly-sized cells (see Table 2.5 for grid dimensions). Each measure is bound to the range $[0, 1]$, the min and max proportion of time that one foot can contact the ground.

Each algorithm evaluates 1 million solutions in the environment. Due to computational limits, we evaluate each solution once instead of averaging multiple episodes, so each algorithm runs 1 million episodes total. Refer to Table 3.2-3.6 for detailed hyperparameters.

3.5.1.3 Metrics

Our primary metric is *QD score* [190], which provides a holistic view of algorithm performance. QD score is the sum of the objective values of all elites in the archive, i.e. $\sum_{i=1}^{M} \mathbf{1}_{\boldsymbol{\phi}_i \text{exists}} f(\boldsymbol{\phi}_i)$, where M is the number of archive cells. We note that the contribution of a cell to the QD score is 0 if the cell is unoccupied. We set the objective f to be the *expected undiscounted return*, i.e. we set $\gamma = 1$ in Eq. 3.2.

Since objectives may be negative, an algorithm's QD score may be penalized when adding a new solution. To prevent this, we define a *minimum objective* in each environment by taking the lowest objective value that was inserted into the archive in any experiment in that environment. We subtract this minimum from every solution, such that every solution that was inserted into an archive has an objective value of at least 0. Thus, we use QD score defined as $\sum_{i=1}^{M} \mathbf{1}_{\boldsymbol{\phi}_i \text{exists}} (f(\boldsymbol{\phi}_i) - \min \text{objective})$. We also define a *maximum objective* equivalent to each environment's "reward threshold" in PyBullet Gym. This threshold is the objective value at which an agent is considered to have successfully learned to walk.

We report two metrics in addition to QD score. *Archive coverage*, the proportion of cells for which the algorithm found an elite, gauges how well the QD algorithm explores measure space, and *best performance*, the highest objective of any elite in the archive, gauges how well the QD algorithm exploits the objective.

3.5.2 Experimental Design

We follow a between-groups design, where the two independent variables are environment (QD Ant, QD Half-Cheetah, QD Hopper, QD Walker) and algorithm (CMA-MEGA (ES), CMA-MEGA (TD3, ES), PGA-ME, ME-ES, MAP-Elites). The dependent variable is the QD score. In each environment, we run each algorithm for 5 trials with different random seeds and test three hypotheses:

H1: CMA-MEGA (ES) will outperform all baselines (PGA-ME, ME-ES, MAP-Elites).

H2: CMA-MEGA (TD3, ES) will outperform all baselines.

Table 3.2: CMA-MEGA (ES) and CMA-MEGA (TD3, ES) hyperparameters. n_{pg} and n_{crit} are only applicable in CMA-MEGA (TD3, ES). n_{pg} here is analogous to n_{pg} in PGA-ME, but we make it much larger here (65,536 vs. 256) to improve the accuracy of the gradient estimate. It is important to obtain a more accurate gradient estimate since we only compute one gradient per iteration instead of taking gradient steps on multiple solutions.

Parameter	Description	Value
N	Iterations = 1,000,000 / $(\lambda + \lambda_{es})$	5,000
λ	Batch size	100
σ_g	Initial CMA-ES step size	1.0
η	Gradient ascent learning rate	1.0
λ_{es}	ES batch size	100
σ_e	ES noise standard deviation	0.02
n_{pg}	TD3 gradient estimate batch size	65,536
n_{crit}	TD3 critic training steps	600

Table 3.3: PGA-ME hyperparameters.

Parameter	Description	Value
N	Iterations = 1,000,000 / λ	10,000
λ	Batch size	100
n_{evo}	Variation operators split	$0.5\lambda = 50$
n_{grad}	PG variation steps	10
α_{grad}	PG variation learning rate (for Adam)	0.001
n_{pg}	PG variation batch size	256
n_{crit}	TD3 critic training steps	300
σ_1	GA variation 1	0.005
σ_2	GA variation 2	0.05
G	Random initial solutions	100

Table 3.4: ME-ES hyperparameters. We adopt the explore-exploit variant.

Parameter	Description	Value
N	Iterations = 1,000,000 / λ	5,000
λ	Batch size	200
σ	ES noise standard deviation	0.02
n_{optim_gens}	Consecutive generations to optimize a solution	10
α	Learning rate for Adam	0.01
α_2	L2 coefficient for Adam	0.005
k	Nearest neighbors for novelty calculation	10

Table 3.5: MAP-Elites hyperparameters. We describe MAP-Elites in Sec. 3.3.2.1.

Parameter	Description	Value
N	Iterations = 1,000,000 / λ	10,000
λ	Batch size	100
σ	Gaussian noise standard deviation	0.02

Table 3.6: TD3 hyperparameters common to CMA-MEGA (TD3, ES) and PGA-ME, which both train a TD3 instance. Furthermore, though we record the objective with $\gamma = 1$ (Sec. 3.5.1.3), TD3 still executes with $\gamma < 1$.

Parameter	Description	Value
—	Critic layer sizes	$[256, 256, 1]$
α_{crit}	Critic learning rate (for Adam)	3e-4
n_q	Critic training batch size	256
$\lvert\mathcal{B}\rvert$	Max replay buffer size	1,000,000
γ	Discount factor	0.99
τ	Target network update rate	0.005
d	Target network update frequency	2
σ_p	Smoothing noise standard deviation	0.2
c_{clip}	Smoothing noise clip	0.5

H3: CMA-MEGA (TD3, ES) will outperform CMA-MEGA (ES).

H1 and H2 are based on prior work [73] which showed that in QD benchmark domains, CMA-MEGA outperforms algorithms that do not leverage both objective and measure gradients. H3 is based on results [178] which suggest that actor-critic methods outperform ES in PyBullet Gym. Thus, we expect the TD3 objective gradient to be more accurate than the ES objective gradient, leading to more efficient traversal of objective-measure space and higher QD score.

3.5.3 Implementation

We implement all QD algorithms with the pyribs library [238] except for ME-ES, which we adapt from the authors' implementation. We run each experiment with 100 CPUs on a high-performance cluster. We allocate one NVIDIA Tesla P100 GPU to algorithms that train TD3 (CMA-MEGA (TD3, ES) and PGA-ME). Depending on the algorithm and environment, each experiment lasts 4-20 hours; refer to Table 3.12 for mean runtimes. We have released our source code at `https://github.com/icaros-usc/dqd-rl`

3.6 Results

We ran 5 trials of each algorithm in each environment. In each trial, we allocated 1 million evaluations and recorded the QD score, archive coverage, and best performance. Fig. 3.4 plots these metrics. Tables 3.7-3.12 show the QD score (Sec. 3.5.1.3), QD score AUC (Sec. 3.6.2.1), archive coverage (Sec. 3.5.1.3), best performance (Sec. 3.5.1.3), mean elite robustness (Sec. 3.6.2.4), and runtime in hours for all algorithms in all environments. The tables show the value of each metric after 1 million evaluations, averaged over 5 trials. Due to its magnitude, QD score AUC is expressed as a multiple of 10^{12}. Finally, Fig. 3.5 and Fig. 3.6 show heatmaps and histograms of the archives from the experiments.

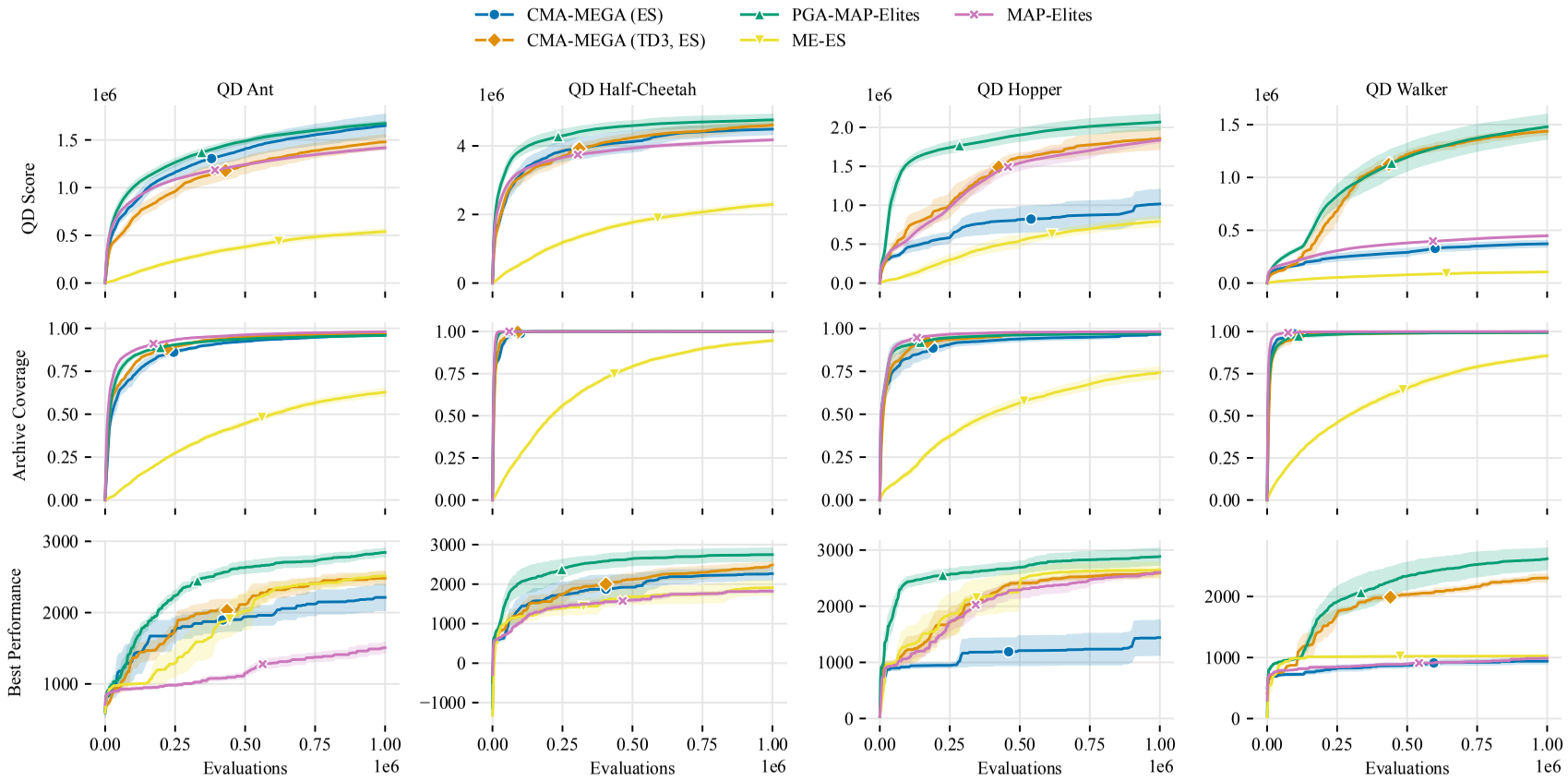


Figure 3.4: Plots of QD score, archive coverage, and best performance for the 5 algorithms in our experiments in all 4 environments from QDGym. The x-axis in all plots is the number of solutions evaluated. Solid lines show the mean over 5 trials, and shaded regions show the standard error of the mean.

Table 3.7: QD Score

	QD Ant	QD Half-Cheetah	QD Hopper	QD Walker
CMA-MEGA (ES)	1,649,846.69	4,489,327.04	1,016,897.48	371,804.19
CMA-MEGA (TD3, ES)	1,479,725.62	4,612,926.99	1,857,671.12	1,437,319.62
PGA-MAP-Elites	1,674,374.81	4,758,921.89	2,068,953.54	1,480,443.84
ME-ES	539,742.08	2,296,974.58	791,954.55	105,320.97
MAP-Elites	1,418,306.56	4,175,704.19	1,835,703.73	447,737.90

Table 3.8: QD Score AUC (multiple of 10^{12})

	QD Ant	QD Half-Cheetah	QD Hopper	QD Walker
CMA-MEGA (ES)	1.31	3.96	0.74	0.28
CMA-MEGA (TD3, ES)	1.14	3.97	1.39	1.01
PGA-MAP-Elites	1.39	4.39	1.81	1.04
ME-ES	0.35	1.57	0.49	0.07
MAP-Elites	1.18	3.78	1.34	0.35

3.6.1 Analysis

To test our hypotheses, we conducted a two-way ANOVA which examined the effect of algorithm and environment on the QD score. We note that the ANOVA requires QD scores to have the same scale, but each environment's QD score has a different scale by default. Thus, for this analysis, we normalized QD scores by dividing by each environment's maximum QD score, defined as *grid cells* * (*max objective - min objective*).

We found a statistically significant interaction between algorithm and environment on QD score, $F(12, 80) = 16.82, p < 0.001$. Simple main effects analysis indicated that the algorithm had a significant effect on QD score in each environment, so we ran pairwise comparisons (two-sided t-tests) with Bonferroni corrections. Our results are as follows:

H1: There is no significant difference in QD score between CMA-MEGA (ES) and PGA-ME in QD Ant and QD Half-Cheetah, but in QD Hopper and QD Walker, CMA-MEGA (ES) attains significantly lower QD score than PGA-ME. CMA-MEGA (ES) achieves significantly higher QD score than ME-ES in all environments except QD Hopper, where there is no significant difference. There is no significant difference between CMA-MEGA (ES) and MAP-Elites in all domains except QD Hopper, where CMA-MEGA (ES) attains significantly lower QD score.

Table 3.9: Archive Coverage

	QD Ant	QD Half-Cheetah	QD Hopper	QD Walker
CMA-MEGA (ES)	0.96	1.00	0.97	1.00
CMA-MEGA (TD3, ES)	0.97	1.00	0.98	1.00
PGA-MAP-Elites	0.96	1.00	0.97	0.99
ME-ES	0.63	0.95	0.74	0.86
MAP-Elites	0.98	1.00	0.98	1.00

Table 3.10: Best Performance

	QD Ant	QD Half-Cheetah	QD Hopper	QD Walker
CMA-MEGA (ES)	2,213.06	2,265.73	1,441.00	940.50
CMA-MEGA (TD3, ES)	2,482.83	2,486.10	2,597.87	2,302.31
PGA-MAP-Elites	2,843.86	2,746.98	2,884.08	2,619.17
ME-ES	2,515.20	1,911.33	2,642.30	1,025.74
MAP-Elites	1,506.97	1,822.88	2,602.94	989.31

H2: In all environments, there is no significant difference in QD score between CMA-MEGA (TD3, ES) and PGA-ME. CMA-MEGA (TD3, ES) achieves significantly higher QD score than ME-ES in all environments. CMA-MEGA (TD3, ES) achieves significantly higher QD score than MAP-Elites in QD Half-Cheetah and Walker, with no significant difference in QD Ant and QD Hopper.

H3: CMA-MEGA (TD3, ES) achieves significantly higher QD score than CMA-MEGA (ES) in QD Hopper and QD Walker, but there is no significant difference in QD Ant and QD Half-Cheetah.

3.6.2 Discussion

We discuss how the CMA-MEGA variants differ from the baselines (Sec. 3.6.2.1-3.6.2.4) and how they differ from each other (Sec. 3.6.2.5).

Table 3.11: Mean Elite Robustness

	QD Ant	QD Half-Cheetah	QD Hopper	QD Walker
CMA-MEGA (ES)	-51.62	-105.81	-187.44	-86.45
CMA-MEGA (TD3, ES)	-48.91	-80.78	-273.68	-97.40
PGA-MAP-Elites	-4.16	-92.38	-435.45	-74.26
ME-ES	77.76	-645.40	-631.32	2.05
MAP-Elites	-109.42	-338.78	-509.21	-186.14

Table 3.12: Runtime (Hours)

	QD Ant	QD Half-Cheetah	QD Hopper	QD Walker
CMA-MEGA (ES)	7.40	7.24	3.84	3.52
CMA-MEGA (TD3, ES)	16.26	22.79	13.43	13.01
PGA-MAP-Elites	19.99	19.75	12.65	12.86
ME-ES	8.92	10.25	4.04	4.12
MAP-Elites	7.43	7.37	4.59	5.72

3.6.2.1 PGA-ME and objective-measure space exploration

Of the CMA-MEGA variants, CMA-MEGA (TD3, ES) performed the closest to PGA-ME, with no significant QD score difference in any environment. This result differs from prior work [73] in QD benchmark domains, where CMA-MEGA outperformed OG-MAP-Elites, a baseline DQD algorithm inspired by PGA-ME.

We attribute this difference to the difficulty of exploring objective-measure space in the benchmark domains. For example, the linear projection benchmark domain is designed to be "distorted" [77]. Values in the center of its measure space are easy to obtain with random sampling, while values at the edges are unlikely to be sampled. Hence, high QD score arises from exploring measure space and filling the archive. Since CMA-MEGA adapts its sampling distribution, it is able to perform this exploration, while OG-MAP-Elites remains "stuck" in the center of the measure space.

In contrast, as discussed in Sec. 3.5.1.1, it is relatively easy to fill the archive in QDGym. We see this empirically: in all environments, all algorithms achieve nearly 100% archive coverage, usually within the first 250k evaluations (Fig. 3.4). Hence, the best QD score is achieved by increasing the objective value of solutions after filling the archive. PGA-ME achieves this by optimizing half of its generated solutions with respect to its TD3 critic. The genetic operator

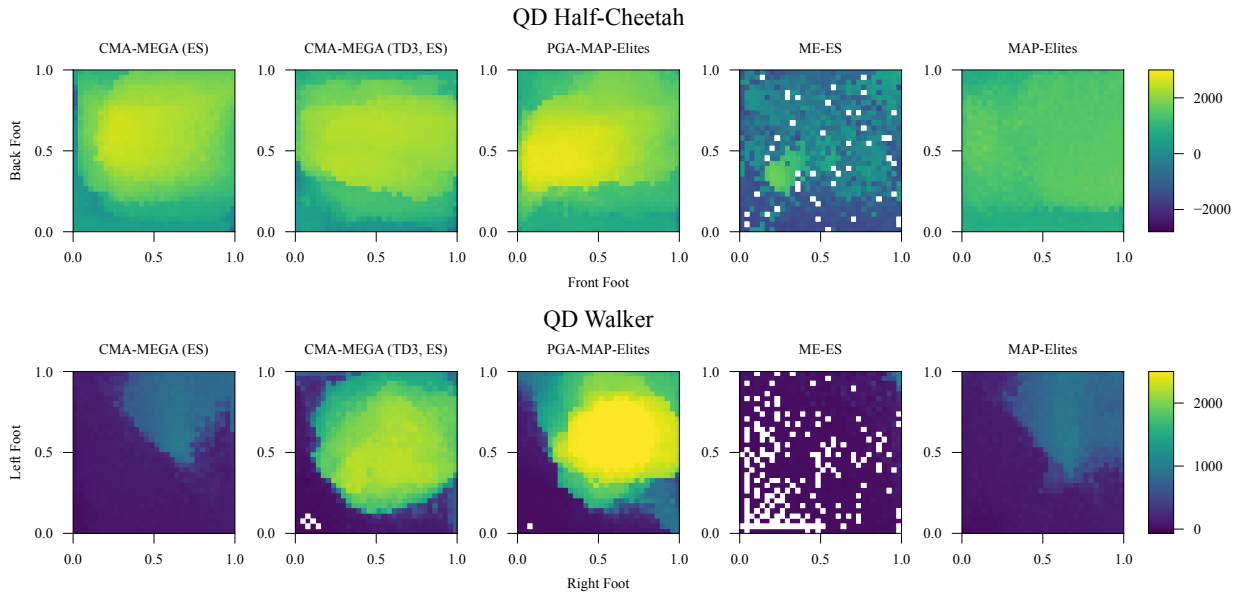


Figure 3.5: Archive heatmaps from the median trial (in terms of QD score) of each algorithm in QD Half-Cheetah and QD Walker. The colorbar for each environment ranges from the minimum to maximum objective stated in Table 3.1. The archive in both environments is a 32×32 grid. Currently, we are unable to plot heatmaps for QD Ant and QD Hopper because their archives are not 2D. These heatmaps have several notable features. First, we can see that MAP-Elites primarily discovers low-performing solutions. Second, from looking at the heatmap videos, we can see that PGA-ME gradually improves the entire archive "all at once" — this happens because PGA-ME samples solutions uniformly from the archive and applies variations to them, so the entire archive appears to improve simultaneously. Finally, again based on the heatmap videos, we see that the CMA-MEGA variants improve the archive with "paintbrush strokes." This happens because the CMA-MEGA variants gradually move the solution point ϕ^* around the archive while generating solutions around it.

likely further enhances the efficacy of this optimization, by taking previously-optimized solutions and combining them to obtain high-performing solutions in other parts of the archive.

On the other hand, the CMA-MEGA variants place less emphasis on maximizing the performance of each solution, compared to PGA-ME: in each trial, PGA-ME takes 5 million objective gradient steps with respect to its TD3 critic, while the CMA-MEGA variants only compute 5k objective gradients, because they dedicate a large part of the evaluation to estimating the measure gradients. This difference suggests a possible extension to CMA-MEGA (TD3, ES) in which solutions are optimized with respect to the TD3 critic before being evaluated in the environment.

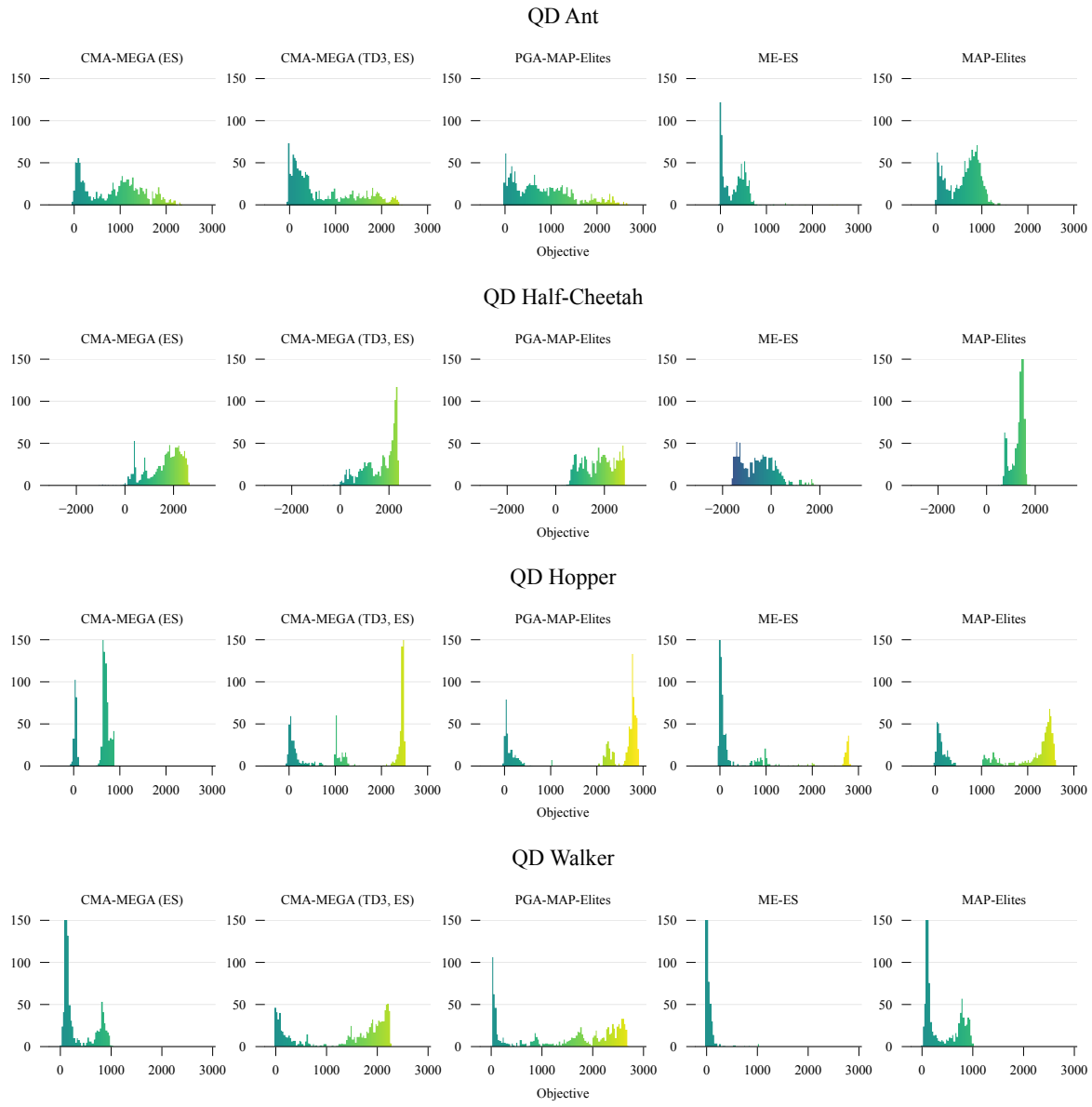


Figure 3.6: Distribution (histogram) of objective values in archives from the median trial (in terms of QD score) of each algorithm in each environment. In each plot, the x-axis is bounded on the left by the minimum objective and on the right by the maximum objective plus 400, as some solutions exceed the maximum objective in Table 2.5. Note that in some plots, the number of items overflows the y-axis bounds (e.g. ME-ES in QD Walker).

3.6.2.2 PGA-ME and optimization efficiency

While there was no significant difference in the *final* QD scores of CMA-MEGA (TD3, ES) and PGA-ME, CMA-MEGA (TD3, ES) was less *efficient* than PGA-ME in some environments. For instance, in QD Hopper, PGA-ME reached 1.5M QD score after 100k evaluations, but CMA-MEGA (TD3, ES) required 400k evaluations.

We can quantify optimization efficiency with *QD score AUC*, the area under the curve (AUC) of the QD score plot. For a QD algorithm which executes N iterations and evaluates λ solutions per iteration, we define QD score AUC as a Riemann sum:

$$\text{QD score AUC} = \sum_{i=1}^{N} (\lambda * \text{QD score at iteration } i) \tag{3.4}$$

After computing QD score AUC, we ran statistical analysis similar to Sec. 3.6.1 and found CMA-MEGA (TD3, ES) had significantly lower QD score AUC than PGA-ME in QD Ant and QD Hopper. There was no significant difference in QD Half-Cheetah and QD Walker. As such, while CMA-MEGA (TD3, ES) obtained comparable final QD scores to PGA-ME in all tasks, it was less efficient at achieving those scores in QD Ant and QD Hopper.

3.6.2.3 ME-ES and archive insertions

With one exception (CMA-MEGA (ES) in QD Hopper), both CMA-MEGA variants achieved significantly higher QD score than ME-ES in all environments. We attribute this result to the number of solutions each algorithm inserts into the archive. Each iteration, ME-ES evaluates 200 solutions but only inserts one into the archive, for a total of 5000 solutions inserted during each run. Given that each archive has at least 1000 cells, ME-ES has, on average, 5 opportunities to insert a solution that improves each cell. In contrast, the CMA-MEGA variants have 100 times more insertions. Though the CMA-MEGA variants evaluate 200 solutions per iteration, they insert 100 of these into the archive. This totals to 500k insertions per run, allowing the CMA-MEGA variants to gradually improve archive cells.

3.6.2.4 MAP-Elites and robustness

In most cases, both CMA-MEGA variants had significantly higher QD score than MAP-Elites or no significant difference, but in QD Hopper, MAP-Elites achieved significantly higher QD score than CMA-MEGA (ES). However, we found that MAP-Elites solutions were less robust.

3.6.2.5 CMA-MEGA variants and gradient estimates

In QD Hopper and QD Walker, CMA-MEGA (TD3, ES) had significantly higher QD score than CMA-MEGA (ES). One potential explanation is that PyBullet Gym (and hence QDGym) augments rewards with reward shaping signals

intended to promote optimal solutions for deep RL algorithms. In prior work [178], these signals led PPO [216] to train successful walking agents, while they led OpenAI-ES into local optima. For instance, OpenAI-ES trained agents which stood still so as to maximize only the reward signal for staying upright.

Due to these signals, TD3's objective gradient seems more useful than that of OpenAI-ES in QD Hopper and QD Walker. In fact, the algorithms which performed best in QD Hopper and QD Walker were ones that calculated objective gradients with TD3, i.e. PGA-ME and CMA-MEGA (TD3, ES).

Prior work [178] found that rewards could be tailored for ES, such that OpenAI-ES outperformed PPO. Extensions of our work could investigate whether there is a similar effect for QD algorithms, where tailoring the reward leads CMA-MEGA (ES) to outperform PGA-ME and CMA-MEGA (TD3, ES).

3.7 Conclusion

To extend DQD to RL settings, we adapted gradient approximations from actor-critic methods and ES. By integrating these approximations with CMA-MEGA, we proposed two novel variants that we evaluated on four locomotion tasks from QDGym. CMA-MEGA (TD3, ES) performed comparably to the state-of-the-art PGA-ME in all tasks but was less efficient in two of the tasks. CMA-MEGA (ES) performed comparably in two tasks.

Our results contrast prior work [73] where CMA-MEGA outperformed a baseline algorithm inspired by PGA-ME in QD benchmark domains. The difference seems to be that difficulty in the benchmarks arises from a hard-to-explore measure space, whereas difficulty in QDGym arises from an objective which requires rigorous optimization. As such, future work could formalize the notions of "exploration difficulty" of a measure space and "optimization difficulty" of an objective and evaluate algorithms in benchmarks that cover a spectrum of these metrics.

For practitioners looking to apply DQD in RL settings, we recommend estimating objective gradients with an off-policy actor-critic method such as TD3 instead of with an ES. Due to the difficulty of modern control benchmarks, it is important to efficiently optimize the objective — TD3 benefits over ES since it can compute the objective gradient without further environment interaction. Furthermore, reward signals in these benchmarks are designed for deep RL methods, making TD3 gradients more useful than ES gradients.

By reducing QD-RL to DQD, we have decoupled QD-RL into DQD optimization and RL gradient approximations. In the future, we envision algorithms which benefit from advances in either more efficient DQD or more accurate RL gradient approximations.

Chapter 4

On-Policy Reinforcement Learning and Differentiable Quality Diversity

4.1 Introduction

* Quality Diversity (QD) algorithms enable the exploration and discovery of diverse skills in a behavior space. For example, a QD algorithm can train different locomotion gaits for a walker [49], discover different grasping trajectories for a manipulator [164], or generate a diverse range of human faces [73]. However, since these algorithms are generally oriented towards solving exploration problems, they struggle to find performant policies in high-dimensional robot learning tasks. QD-RL is an emerging field that attempts to combine the principled exploration capabilities of QD with the powerful performance improvement capabilities of RL. Prior methods have leveraged *off-policy* RL, specifically TD3, to estimate the gradient of performance, and either Evolution Strategies (ES) or TD3 to estimate the gradient of diversity in order to search for diverse, high-quality policies. They have shown success in exploration problems and certain robot locomotion tasks [175, 186, 239]. Nonetheless, there remains a gap in performance between QD-RL and standard RL algorithms on continuous control tasks. Furthermore, off-policy RL algorithms were not designed with massive parallelization in mind. Few works explore how to leverage modern massively-parallelized simulators with these algorithms, whereas numerous works explore on-policy RL in these regimes [155, 201, 107, 13, 117].

From our investigation of prior methods, simply combining existing QD methods with an RL algorithm tends not to scale well to high-dimensional, highly dynamical systems such as Humanoid. For example, all contemporary QD-RL algorithms for locomotion use non-Markovian measures of behavioral diversity, which in many cases prevents direct RL-optimization. Most algorithms instead opt for policy parameter mutation, which struggles to scale well with deep

*Work led by Sumeet Batra at the Robotic Embedded Systems Laboratory.

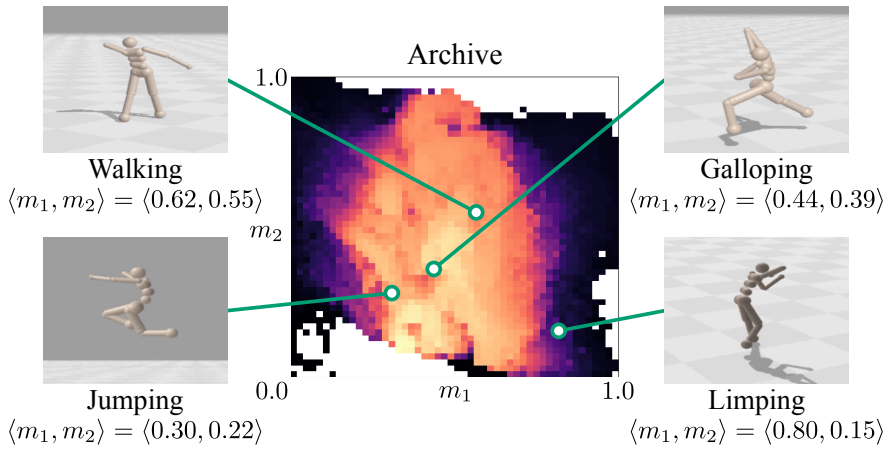


Figure 4.1: PPGA finds a diverse archive of high-performing locomotion behaviors for a humanoid agent by combining PPO gradient approximations with Differentiable Quality Diversity algorithms. The archive's dimensions correspond to the measures m_1 and m_2, i.e., the proportion of time that the left and right feet contact the ground. The color of each cell shows the objective value, i.e., how fast the humanoid moves. For instance, jumping moves the humanoid forward quickly, with the left and right feet individually contacting the ground 30% and 22% of the time, respectively.

neural networks. Prior methods that investigated combining Differentiable Quality Diversity and *off-policy* RL [239] achieved similar results as other baselines. However, given the gap in performance between standard RL and QD-RL algorithms in terms of best-performing policy, we believe that DQD algorithms, under a different formulation more synergistic with its underlying mechanisms, can close this gap. To this end, we leverage Proximal Policy Optimization (PPO) [216], a popular *on-policy* RL algorithm, with Differentiable Quality Diversity (DQD) [73] because of the already present synergy. Specifically, DQD algorithms CMA-MEGA [73], and its more recent variation CMA-MAEGA [72], maintain a single search point (or policy in the case of RL) that moves through the behavior space and fills in new, unexplored regions with offspring policies constructed via gradient information *collected from online data*. It is through this high level view that we see the emergent synergy between PPO and DQD, in that PPO can be used to collect gradient estimates from online data when one or both of the objective and measure functions are Markovian and non-differentiable.

We make several key changes to CMA-MAEGA and PPO to maximally leverage their synergy. Our new algorithm, Proximal Policy Gradient Arborescence (PPGA), to the best of our knowledge, is the first QD-RL algorithm to not only achieve 4x performance in best reward on the humanoid domain, but achieve the *same* level of performance as PPO without sacrificing *any* of the diversity in the discovered policies. Specifically, we make the following contributions:

(1) We propose a vectorized implementation of PPO, VPPO, that jointly computes the objective and measure gradients with little overhead and without running separate PPO instances for each task **(2)** We generalize prior

CMA-based DQD algorithms as instances of Natural Evolution Strategies (NES) and show that contemporary NES methods, specifically xNES, enable better training stability and performance for DQD algorithms **(3)** We introduce the notion of Markovian Measure Proxies (MMPs), which makes the typically non-Markovian measure functions used in QD-RL amenable to RL-optimization **(4)** We propose a new method to move the current search point, hereon referred to as the "search policy," to unexplored regions of the archive by iteratively "walking" it using collected online data and RL optimization of a novel multi-objective reward function.

4.2 Background

4.2.1 Deep Reinforcement Learning

Reinforcement Learning algorithms search for a policy, a mapping of states to actions, that maximizes cumulative reward in an environment. RL assumes the discrete-time Markov Decision Process (MDP) formalism $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$ where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces respectively, $\mathcal{R}(s, a)$ is the reward function, $\mathcal{P}(s'|s, a)$ defines state transition probabilities, and γ is the discount factor. The RL objective is to maximize the discounted episodic return of a policy $\mathbb{E}\left[\sum_{k=0}^{T-1} \gamma^k R(s_k, a_k)\right]$ where T is episode length. **Deep Reinforcement Learning** solves the RL problem by finding a policy $\pi_\theta(a_t|s_t)$ parameterized by a deep neural network θ that represents a state-action mapping.

On-policy Deep RL methods directly learn the policy π_θ using experience collected by that policy or a recent version thereof. Contemporary methods [160] fit the value function $V_\phi(s_t)$ to discounted returns and estimate the advantage $\hat{A}_t = \sum_{k=t}^{T-1} \gamma^k R(s_k, a_k) - V_\phi(s_t)$, which corresponds to the value of an action over the current policy [215]. From here, the gradient of the objective w.r.t. θ, or **policy gradient**, can be estimated as $\hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}_t\right]$, and the policy π_θ is trained using mini-batch gradient descent.

Trust region policy gradient Deep RL methods constrain the policy updates to maintain the proximity of π_θ to the *behavior* policy $\pi_{\theta_{old}}$ that was used to collect the experience. TRPO [214] takes the largest policy improvement step that satisfies the strict KL-divergence constraint. Proximal Policy Optimization (PPO) [216] approximates the trust region by optimizing a clipped surrogate objective where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the importance sampling ratio:

$$L(\theta) = \hat{\mathbb{E}}_{\pi_\theta} \left[\min(r_t(\theta)\hat{A}_t), \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right].$$

Off-policy Deep RL algorithms learn parameterized state-action value functions $Q_\theta(s_t, a_t)$ that estimate the value of taking action a_t in state s_t. Then, actions are taken with a greedy policy $\arg\max_a Q_\theta(s_t, a_t)$, or an ε-greedy variation thereof. Q-functions can be learned from experience collected by recent or past versions of the policy or another policy altogether.

In continuous control problems, it can be difficult to find $a^* = \arg\max_a Q_\theta(s_t, a_t)$ due to an infinite number of possible actions. To work around this issue, off-policy methods such as DDPG [149] and TD3 [81] learn a deterministic policy $\mu_\phi(s_t)$ by solving $max_\phi(Q_\theta(s_t, \mu_\phi(s_t))$ using gradient ascent. Other off-policy methods, such as soft actor-critic (SAC) [103], maintain an explicit policy π_θ, but similarly derive the policy gradient from the critic, allowing them to learn from off-policy data as well.

4.2.2 Quality Diversity Optimization

Unlike single-objective optimization methods such as RL, Quality Diversity algorithms search for an archive of high-performing, diverse policies. An optimal archive essentially answers the question, "how does performance change with behavior?" by mapping out the optimization landscape of a pre-defined behavior space. The QD problem [35] assumes an *objective* function $f(\cdot)$ that quantifies the agent's performance and k *measure* functions $m_1(\cdot)...m_k(\cdot)$ that characterize the agent's behavior. The measure functions, represented jointly as $\mathbf{m}(\cdot)$, define an embedding the QD algorithm should span with diverse policies. The **QD objective** is to find a policy that maximizes f for every possible output of $\mathbf{m}$. However, the embedding formed by $\mathbf{m}$ is continuous, so the embedding space is discretized into a tessellation of M cells. The QD objective then becomes to maximize $\sum_{i=1}^{M} f(\theta_i)$, where θ_i is a policy whose measures $\mathbf{m}(\theta_i)$ fall in cell i of the tesselation.

QD algorithms originated with NSLC [145, 146] and MAP-Elites [169, 49]. While these early QD algorithms built on genetic algorithms, modern QD algorithms incorporate optimization techniques like evolution strategies [77, 44, 42], gradient ascent [73, 72], and differential evolution [38]. Several works have applied QD optimization to generative design [105, 83], procedural content generation [95, 61, 135], robot manipulation [165], and reinforcement learning [175, 239].

4.2.3 Differentiable Quality Diversity

The Differentiable Quality Diversity (DQD) [73] algorithm Covariance Matrix Adaptation Map Elites via Gradient Arborescence (CMA-MEGA) considers the first-order QD problem where the objective and measure functions are differentiable, with gradients w.r.t. policy parameters represented as $\nabla f = \frac{\partial f}{\partial \theta}$ and $\nabla \mathbf{m} = \left[\frac{\partial m_1}{\partial \theta}, ..., \frac{\partial m_k}{\partial \theta}\right]$. CMA-MEGA maintains a *search policy* π_{θ_μ} in policy parameter space ($\theta_\mu \in \mathbb{R}^N$) corresponding to some cell in the archive given by the measures $< m_1(\pi_{\theta_\mu}), ..., m_k(\pi_{\theta_\mu}) >$, and a search distribution in objective-measure gradient coefficient space maintained by CMA-ES [108], a zeroth-order optimizer that optimizes the coefficient distribution to produce coefficient vectors that point in the direction of *greatest archive improvement*. At a high level, CMA-MEGA branches off policies from the search policy in order to locally fill the archive, and then steps the search policy to new, unexplored regions of the archive. During the branching step, the gradients $< \nabla f, \nabla \mathbf{m} >_{\theta_\mu}$ and λ gradient coefficient vectors $< c_0, ..., c_k >_1, ..., < c_0, ..., c_k >_\lambda$ sampled from the CMA-ES search distribution $\mathbf{c}_i \sim \mathcal{N}(\mu, \Sigma) \in \mathbb{R}^{k+1}$ are combined via the dot product i.e. $< \nabla f, \nabla \mathbf{m} >_{\theta_\mu} \cdot \mathbf{c}_1, ...$ to produce local gradients $\nabla_1, ..., \nabla_\lambda$ around the search policy. Applying the gradients to the search policy gives us λ branched policies $\pi_{\theta_1}, ..., \pi_{\theta_\lambda}$. The new policies can then be ranked by how much they improve the archive, i.e., $f(\pi_{\theta_i}) - f(\pi_{\theta_{old}}), i \in [1, \lambda]$, where $\pi_{\theta_{old}}$ is the incumbent policy in the archive corresponding to the same cell as π_{θ_i}. Branched policies that map to new, unexplored cells in the archive have $f(\pi_{\theta_{old}})$ set to some minimum threshold. This implicitly biases the ranking towards the exploration of new, unvisited cells. This ranking is given to CMA-ES, which internally performs an update that steps the search distribution in the direction of the natural gradient w.r.t. greatest archive improvement. CMA-ES returns weights $w_1, ..., w_\lambda$ such that $\nabla_{step} =< w_1, ..., w_\lambda > \cdot < \nabla_1, ..., \nabla_\lambda >$ is the natural gradient in parameter space. This weighted linear recombination of the branching gradients is then used to step the search policy in the direction of greatest archive improvement $\theta_\mu \leftarrow \theta_\mu + \alpha \nabla_{step}$.

The current state-of-the-art DQD algorithm, Covariance Matrix Adaptation Map Annealing via Gradient Arborescence (CMA-MAEGA) [72], introduced the concept of soft archives to CMA-MEGA. Instead of maintaining the best policy in each cell, the archive maintains a threshold t_e and updates the threshold by $t_e \leftarrow (1 - \alpha)t_e + \alpha f(\pi_{\theta_i})$ when a new policy π_{θ_i} crosses the threshold of its cell e. The hyperparameter $0 \leq \alpha \leq 1$, referred to as the *archive learning rate*, controls how much time is spent optimizing a region of the archive before exploring a new region. Soft archives

have many theoretical and practical benefits discussed in prior work [72]. Our proposed PPGA algorithm builds directly on CMA-MAEGA.

4.2.4 Quality Diversity Reinforcement Learning

Unlike the standard DQD formulation in which the analytical gradients of f and $\mathbf{m}$ can be computed, the QD-RL setting considers MDPs in which these functions are non-differentiable and must be *approximated* with model-free RL. The gradient approximations of f and $\mathbf{m}$ can be used to improve the performance and diversity of agents in an archive. QD-RL methods can be roughly divided into two subgroups. The first set of approaches directly optimizes over the entire archive by sampling existing policies in the archive and applying operations to the policies' parameters that either improve their performance or diversity. For example, PGA-ME [175] collects experience from evaluated agents into a replay buffer and uses TD3 to derive a policy gradient that improves the performance of randomly sampled agents from the archive, while using genetic variation [249] on the same set of agents to improve diversity and fill new, unexplored cells. Similarly, QDPG [186] derives a policy and diversity gradient using TD3 and applies these operators to randomly sampled agents in the archive.

Whereas the first family of QD-RL algorithms simultaneously search the behavioral embedding in many different regions at once, the second family uses the DQD formulation i.e., maintains a *single search policy* that explores new local regions one at a time using objective-measure gradient *approximations*. In prior work [239], the authors considered objective gradient approximations via TD3 and OpenAI-ES, while approximating the measure function gradients with OpenAI-ES. In this work, we notice the unique on-policy nature of DQD algorithms and present a novel formulation that exploits their synergy with PPO.

4.3 Proposed Method: The Proximal Policy Gradient Arborescence Algorithm

We begin with the DQD algorithm CMA-MAEGA as our foundation. The algorithm can be roughly divided into three phases: (1) computing the objective-measure gradients for the branching phase, (2) providing the relative ranking of each branched policy w.r.t. the QD objective to CMA-ES, and (3) stepping the search policy in the direction of greatest archive improvement. Sec. 4.3.1 and Sec. 4.3.2 focus on enabling RL optimization for phase one, Sec. 4.3.3 explores

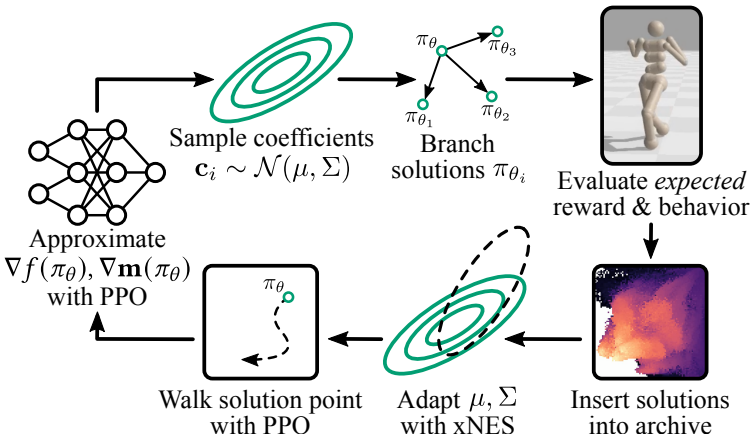


Figure 4.2: PPGA estimates $\nabla f, \nabla \mathbf{m}$ with PPO. We randomly sample gradient coefficients $\mathbf{c}$ and perform weighted linear recombination of the objective-measure gradients with $\mathbf{c}$ as the weights. This produces a population of gradients that, in turn, result in a population of branched policies. The policies are evaluated and inserted into the archive. xNES adapts the gradient coefficient distribution based on these insertions towards maximal archive improvement. The new mean of the coefficient distribution is used to walk the search policy towards a new, potentially unexplored region of the archive.

the connection between CMA-ES and NES and how this can improve training stability in phase two, and Sec. 4.3.4 describes our method for walking the search policy with PPO in phase three.

4.3.1 Markovian Measure Proxies

QD problems often contain non-Markovian measure functions. For robot locomotion tasks, the standard measure function is the proportional foot contact time with the ground $m_i(\theta) = \frac{1}{T}\sum_{t=0}^{T} \delta_i(s_t)$ for each leg $i, i = 1...k$, where the Kronecker delta $\delta_i(s_t)$ indicates whether the i'th leg is in contact with the ground or not in state s_t, and T is the episode length. However, this measure function is defined on a trajectory (i.e., the whole episode), making it non-Markovian and thus preventing us from using RL to estimate its gradient. To solve this issue, we introduce the notion of a **Markovian Measure Proxy** (MMP), which is a surrogate function that obeys the Markov property and has a positive correlation with the original measure function. For locomotion tasks, we can construct an MMP by simply removing the dependency on the trajectory and making the original measure function state-dependent, i.e., setting it to be $\delta_i(s_t)$. We can then use the exact same MDP as the standard RL formulation and replace the reward function with $\delta_i(s_t)$.

4.3.2 Policy Gradients for Differentiable Quality Diversity Optimization

PPO is an attractive choice as our objective-measure gradient estimator because of its ability to scale with additional parallel environments. Being an approximate trust region method, the constrained policy update step provides some robustness to noisy and non-stationary objectives. This is particularly important in the QD-RL setting, where the QD-objective is highly non-stationary – that is, the QD-objective changes with the state of the archive, which is updated on each QD iteration.

We treat the RL objective and k MMPs, each one optimized by an actor-critic pair, as reward functions to optimize. Rather than spawning a new PPO instance with a separate actor and critic network for the RL objective f and each MMP $\delta_i(s_t)$ independently, we start with a single actor $\pi_{\theta_\mu}(a|s)$ parameterized by the policy parameters θ_μ of the current search policy, and $k+1$ value functions $V_{\phi_f}, V_{\phi_{\delta_1}}, ..., V_{\phi_{\delta_k}}$. The actor is replicated $k+1$ times, each one paired with a corresponding value function. The actors are combined into a single *vectorized policy* $\pi_{<\theta_\mu^1,...,\theta_\mu^{k+1}>}(a|s)$ that jointly optimizes $< f, \delta_1(s_t), ..., \delta_1(s_t) >$ for N_1 iterations, where N_1 is a configurable hyperparameter. We additionally modify the computation of the policy gradient into a *batched policy gradient* method, where intermediate gradient estimates of each function w.r.t. policy params only flow back to the parameters corresponding to respective individual policies during minibatch gradient descent. After N_1 iterations, we separate the vectorized policy, giving a set of subpolicies with optimized parameters $\pi_{\theta_f}(a|s), ..., \pi_{\theta_{\delta_k}}(a|s)$ that perform better w.r.t. their objectives. In the case of measure functions where $m_i(\cdot)$ is the proportion foot contact time of the $i'th$ leg, $m_i(\pi_{\theta_{\delta_k(s_t)}}) > m_i(\pi_{\theta_\mu})$ i.e. the resulting policy will have a higher proportion foot contact time over the starting policy after optimization. Subtracting the initial parameters (θ_μ) from each resulting policy gives us the desired objective-measure Jacobian $\left[\frac{\partial f}{\partial \theta_\mu}, \frac{\partial \delta_1}{\partial \theta_\mu}, ..., \frac{\partial \delta_k}{\partial \theta_\mu}\right]$, which can be linearly recombined in various ways to branch policies from θ_μ.

In addition to the VPPO implementation, we introduce the option to make the learnable action standard deviation parameter static. In the typical case, PPO decays this parameter over time in order to converge to a quasi-deterministic optimal policy at the expense of further exploration. In some environments, narrowing the action distribution can indeed help promote consistent optimal performance. In other environments, this effect can hinder the QD algorithm's ability to branch policies into new cells, given that the outer QD optimization loop relies on gradient estimates produced by PPO to discover unexplored regions of the archive. In environments where we observe this negative effect, we disable gradient flow to the action standard deviation parameter.

Finally, in order to address environmental uncertainty, we insert new policies based on their performance and behavior *averaged* over 10 parallel environments. We leverage GPU acceleration to quickly batch process many parallel environments over a population of branched policies.

4.3.3 Connection to Natural Evolution Strategies

We replace CMA-ES with a Natural Evolution Strategy (NES) to increase the stability and performance of CMA-MAEGA on noisy RL environments. CMA-based variants of PPGA diverged during training. Prior work [171] showed that CMA-ES struggled to evolve deep neural network controllers with dimensionality $\mathbb{R}^d$ on stochastic RL environments. However, CMA-MAEGA uses CMA-ES to maintain search distribution in objective-measure gradient coefficient space $\mathbb{R}^{k+1} << \mathbb{R}^d$, where $k+1$ can be as small as three dimensions, implying that CMA-ES should still be effective in this low-dimensional space. It was then puzzling to find consistent divergence during the training of our CMA-based algorithm. We hypothesize that the culprit is the cumulative step-size adaptation (CSA) mechanism employed by CMA-ES. CMA-ES uses evolution paths $\rho_\sigma^{(g)}$ to adapt the step size $\sigma^{(g)}$ between successive generations (g). The mechanisms by which $\sigma^{(g)}$ are updated assume a fairly non-noisy and stationary objective f. However, the application of CMA-ES to QD optimization on stochastic RL environments presumes the exact opposite. That is, the RL objective f_{RL} is very noisy, and the QD-objective $f_{QD} = g(f_{RL}(\cdot))$, which is a function of the RL objective, is highly non-stationary, since the state of the archive $\mathcal{A}$ *changes* the direction of greatest archive improvement on every iteration. To address the training divergence, we propose using exponential evolution strategies (xNES) [90], a more recent and theoretically well-motivated method, as a drop in replacement for CMA-ES. Prior works have shown strong links between xNES and CMA-ES, and generalize both methods as instances of *natural evolution strategies* [4, 90]. In fact, the update step in xNES is equivalent to CMA-ES up to the use of evolution paths. We refer to these prior works for an in-depth comparison. More generally, we believe *any* natural evolution strategy can be used to maintain and update the search distribution over gradient coefficients in this and any prior CMA-based DQD method.

4.3.4 Walking the Search Policy

In standard DQD, ∇_{step} is computed via weighted linear recombination to produce a gradient vector that steps the search policy in the least explored direction of the archive. However, the resulting gradient vector is a linearized

Algorithm 6: Proximal Policy Gradient Arborescence

Input: Initial policy θ_0, VPPO instance to approximate $\nabla f, \nabla \mathbf{m}$ and move the search policy, number of QD iterations N_Q, number of VPPO iterations to estimate the objective-measure functions and gradients N_1, number of VPPO iterations to move the search policy N_2, branching population size λ, and an initial step size for xNES σ_g

Initialize the search policy $\theta_\mu = \theta_0$. Initialize NES parameters $\mu, \Sigma = \sigma_g I$
for iter $\leftarrow 1$ **to** N **do**
  $f, \nabla f, \mathbf{m}, \nabla \mathbf{m} \leftarrow VPPO.compute\_jacobian(\theta_\mu, f(\cdot), \mathbf{m}(\cdot), N_1)$
  $\nabla f \leftarrow$ normalize$(\nabla f), \nabla \boldsymbol{m} \leftarrow$ normalize$(\nabla \boldsymbol{m})$
  $\_ \leftarrow$ update _archive$(\theta_\mu, f, \boldsymbol{m})$
  for $i \leftarrow 1$ **to** λ **do**
    $c \sim \mathcal{N}(\mu, \Sigma)$ // sample gradient coefficients
    $\nabla_i \leftarrow c_0 \nabla f + \sum_{j=1}^{k} c_j \nabla m_j$
    $\theta_i^{'} \leftarrow \theta_\mu + \nabla_i$
    $f', *, m', * \leftarrow$ rollout(θ_i')
    $\Delta_i \leftarrow$ update_archive$(\theta_i^{'}, f', \boldsymbol{m'})$
  end for
  rank gradient coefficients ∇_i by archive improvement Δ_i
  Adapt xNES parameters $\mu = \mu', \Sigma = \Sigma'$ based on improvement ranking Δ_i
  $f'(\theta_\mu) = c_{\mu,0} f + \sum_{j=1}^{k} c_{\mu,j} m_j$, where $\mathbf{c}_\mu = \mu'$ // construct multi-objective reward function
  $\theta'_\mu = VPPO.train(\theta_\mu, f', N_2)$ // standard PPO training procedure
  if *there is no change in the archive* **then**
    Restart xNES with $\mu = 0, \Sigma = \sigma_g I$
    Set θ_μ to a randomly selected existing cell θ_i from the archive
  end if
end for

approximation around the current search policy θ_μ and thus cannot be reused to take multiple gradient steps in a non-convex optimization problem. It would be remiss not to leverage the highly-parallelized VPPO implementation to "walk" the search policy over *many* steps in the direction of greatest archive improvement. We make the key observation that the mean gradient coefficient vector $\mathbf{c}_\mu$ of the *updated* search distribution maintained by xNES points in the direction of greatest archive improvement for the next iteration of the QD algorithm. Thus, we construct a new multi-objective reward function for VPPO to optimize by taking the dot product between the gradient coefficient vector and the objective and measure proxies $< c_{\mu_0}, ..., c_{\mu_{k+1}} > \cdot < f, \delta_1, ..., \delta_k >$. Optimizing this function with VPPO allows us to walk the search policy θ_μ in the direction of greatest archive improvement by iteratively taking conservative steps, where the magnitude of the movement is controllable by hyperparameter N_2. This objective is stationary for all N_2 steps, and is only updated after the subsequent QD iteration. We provide a diagram of our method in Fig. 4.2, and pseudocode in Algorithm 6, Algorithm 7, and Algorithm 8.

Algorithm 7: Update Archive

Input: Solution θ to insert, episodic reward f, measures $\mathbf{m} =< m_1, ..., m_k >$, archive $\mathcal{A}$, archive learning rate α

θ_{inc}, f_{inc} = $\mathcal{A}$[**m**] if $\mathcal{A}$[**m**] is nonempty else $None, 0$ // incumbent policy

$\Delta_i = 0$

if $f > f_{inc}$ **then**

  insert θ into cell $\mathcal{A}$[**m**]

  $f_{inc} \leftarrow (1 - \alpha) f_{inc} + \alpha f$

  $\Delta_i = f - f_{inc}$

end if

return Δ_i

Algorithm 8: Vectorized-PPO (VPPO)

Input: Initial search policy π_{θ_i}, objective functions to optimize $\mathbf{f} = f_1(\cdot), ..., f_k(\cdot)$, number of VPPO iterations N, number of parallel environments E, rollout length L

Initialize the vectorized agent $\mathbf{v}\pi_{\theta_\mathrm{i}}$ = vectorized_agent($[\pi_\theta] \times (k + 1)$)

for iter $\leftarrow 1$ **to** N_1 **do**

  (S, A, R, S') $\leftarrow$ rollout(vectorized_agent, E, L, **f**) // Note that **S** = $\{S_1, ..., S_k\}$, etc

  advantage $\mathbb{A}$, returns $\mathcal{G} \leftarrow$ batch_calculate_rewards(**S, A, R, S'**, **f**)

  $\mathbf{v}\pi_\theta{}' \leftarrow$ batch_gradient_descent($\mathbb{A}, \mathcal{G}, \mathbf{v}\pi_\theta$) // using the stochastic policy gradient

  $\mathbf{v}\pi_\theta \leftarrow \mathbf{v}\pi_\theta{}'$

end for

$\nabla\mathbf{f} \leftarrow \mathbf{v}\pi_\theta{}' - \mathbf{v}\pi_{\theta_\mathrm{i}}$

return $\nabla\mathbf{f}$

4.4 Experiments

We evaluate our algorithm on four different continuous-control locomotion tasks derived from the original Mujoco environments [242]: Ant, Walker2d, Half-Cheetah, and Humanoid. The standard objective in each task is to maximize forward progress and robot stability while minimizing energy consumption. We use the Brax simulator to leverage GPU acceleration and massive parallelization of the environments. The observation space sizes for these environments are 87, 17, 18, and 227, respectively, and the action space sizes are 8, 6, 6, and 17, respectively. The standard Brax environments are augmented with wrappers that determine the measures of an agent in any given rollout as implemented in QDax [150], where the number of measures of an agent is equivalent to the number of legs. The measure function is the number of times a leg contacts the ground divided by the length of the trajectory. We implement PPGA in pyribs [238], with our VPPO implementation based on CleanRL's implementation of PPO [117]. Most experiments were run on a SLURM cluster where each job had access to an NVIDIA RTX 2080Ti GPUs, 4 cores from a Intel(R) Xeon(R) Gold 6154 3.00GHz CPU, and 108GB of RAM. Some additional experiments and ablations were run on local workstations with access to an NVIDIA RTX 3090, AMD Ryzen 7900x 12 core CPU, and 64GB of RAM.

Table 4.1: List of relevant hyperparameters for PPGA shared across all environments.

HYPERPARAMETER	VALUE
ACTOR NETWORK	[128, 128, ACTION DIM]
CRITIC NETWORK	[256, 256, 1]
N_1	10
N_2	10
PPO NUM MINIBATCHES	8
PPO NUM EPOCHS	4
OBSERVATION NORMALIZATION	TRUE
REWARD NORMALIZATION	TRUE
ROLLOUT LENGTH	128

4.4.1 Comparisons

We compare our results to current state-of-the-art QD-RL algorithms: Policy Gradient Assisted MAP-Elites (PGA-ME), Quality Diversity Policy Gradient (QDPG) implemented in QDax [150], and CMA-MAEGA (TD3, ES) implemented in pyribs [238]. We also compare against the state-of-the-art ES-based QD-RL algorithm, separable CMA-MAE (sep-CMA-MAE) [240], which allows evolutionary QD techniques to scale up to larger neural networks. Finally, in order to verify our hypothesis on the emergent synergy between PPO and DQD, we provide an ablation where TD3 is used as a drop-in replacement for PPO in PPGA, which we will refer to as TD3GA going forward. The same archive resolutions and network architectures are used for all baselines. A full list of shared hyperparameters is in Table 4.1. We use an archive learning rate of 0.1, 0.15, 0.1, and 1.0 on Humanoid, Walker2d, Ant, and Half-Cheetah, respectively. Adaptive standard deviation is enabled for Ant and Humanoid. We reset the action distribution standard deviation to 1.0 on each iteration in all other environments.

We conduct our experiments using the following criteria: **QD-score**, which is the sum of scores of all nonempty cells in the archive, and **coverage**, which is the percentage of nonempty cells in the archive, have been historically used by QD algorithms to measure performance and diversity respectively, and so we include them as metrics. However, these metrics have a number of edge cases that make them imperfect measures of performance and diversity. For example, an algorithm that fills 100% of the archive with low-performing policies can have a higher QD-score and coverage than a QD algorithm that fills fewer cells with high-performing policies. To more accurately represent the performance and diversity of a given algorithm, we additionally include plots of the **Complementary Cumulative Distribution Function** (CCDF), originally presented in [248], which shows what percentage of policies in the archive

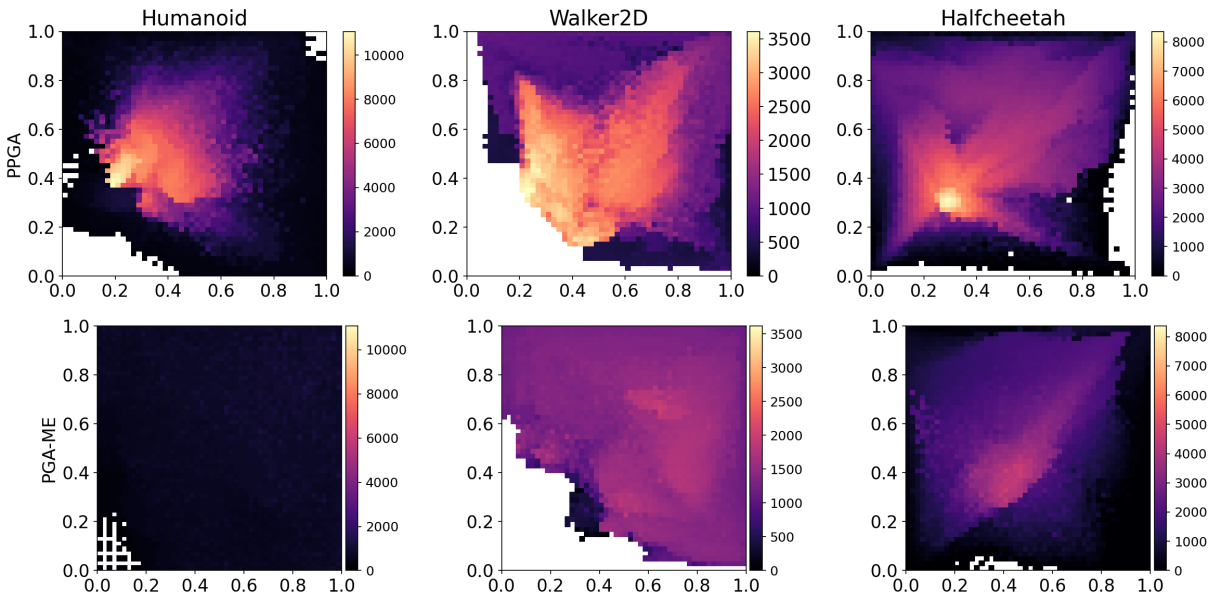


Figure 4.3: 2D Archive visualizations of PPGA compared to the current state-of-the-art QD-RL algorithm PGA-ME. We use 50x50 archives to show detail.

achieve a reward of R or greater for all possible values of R on the x-axis. The CCDF attempts to capture notions of quality of policies in the archive and diversity, while also shedding light on how the policies are *distributed* w.r.t. performance. Finally, we include the **best reward** metric, denoting the highest-performing policy the algorithm was able to discover.

Figures 4.3 and 4.4 show that PPGA outperforms baselines in best reward and QD-score, achieving comparable coverage scores on all tasks except Ant, and generating much more illuminated archive heatmaps with a diverse range of higher performing policies than the current state of the art, PGA-ME. Notably, PPGA is the only algorithm that solves Humanoid, achieving over 4x improvement in best-performing policy and QD score compared to baselines. More important than QD-Score and Coverage are the CCDF plots. At $x = 0$, all policies in the archive are included, i.e., $x = 0$ encapsulates the coverage score. CCDF plots provide a better representation of "quality" than QD-score, since we can see how the policies in the archive are distributed. Except for Ant, PPGA produces distributions where more of the mass is distributed to the right where the high-performing policies lie.

In Figure 4.5, we find evidence that PPO has an important synergy with DQD that is perhaps missing in other RL algorithms. TD3GA fails to find high performing policies on Humanoid. Achieving 100% coverage is indicative of the step size σ in xNES exploding and producing highly stochastic policies that, by chance, land in far away cells.

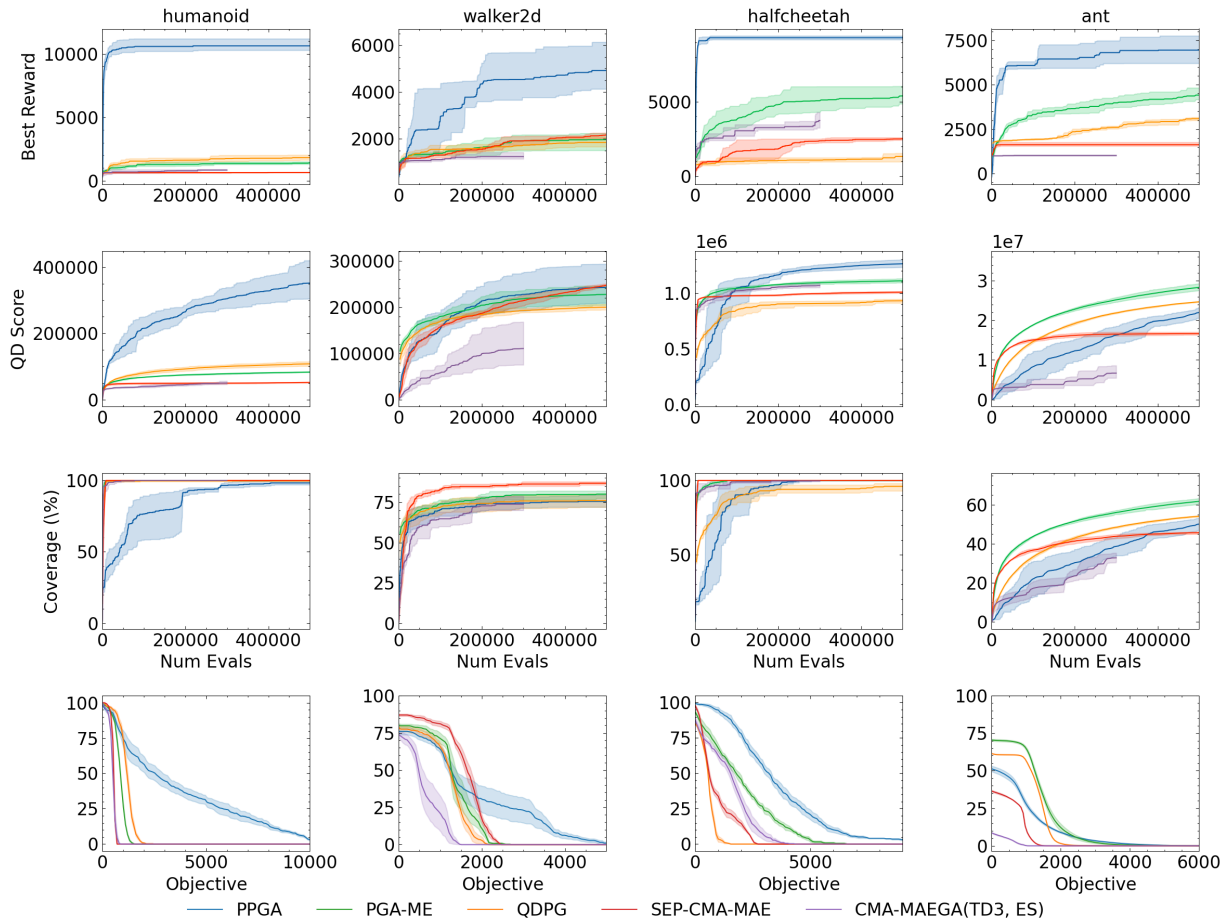


Figure 4.4: QD metrics and cumulative distributions for archives of PPGA and the baselines. The CCDF plots in the last row indicate the percentage of archive policies above a certain objective threshold. All plots show the mean over four seeds with a 95% bootstrapped confidence interval.

This typically occurs when xNES cannot fit a covariance matrix to the data, which in this case are weighted linear combinations of $\nabla f, \nabla \mathbf{m}$ produced by TD3.

4.4.2 Post-Hoc Archive Analysis

QD algorithms are known to struggle with reproducing performance and behavior in stochastic environments. To determine the replicability of our agents, we follow the guidelines of prior work [67]. We re-evaluate each agent in the archive 50 times and average its performance and measures to construct a Corrected Archive and use this to produce Corrected QD metrics such as QD-Score and Coverage. Fig. 4.6 shows the corrected QD metrics and the corrected CCDFs, respectively. After re-evaluation, PPGA maintains the lead in best reward on all tasks, QD-score on

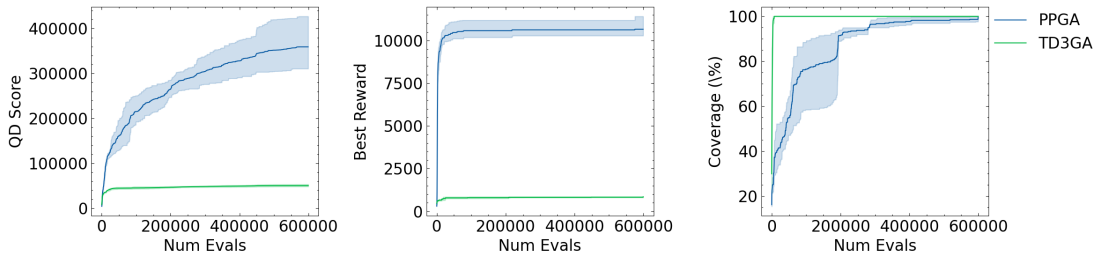
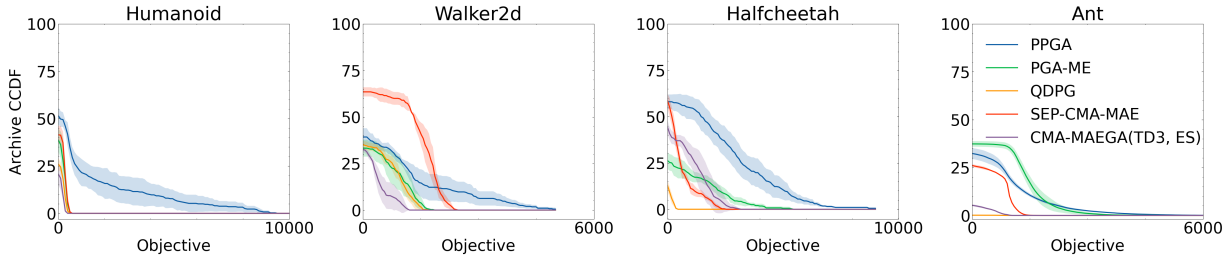


Figure 4.5: PPGA vs TD3GA on Humanoid on the standard QD metrics. All plots are averaged over 4 seeds. The shaded regions are the 95% bootstrapped confidence intervals.



	Humanoid			Walker2d			Halfcheetah			Ant		
	QD-Score	Cov	Best	QD-Score	Cov	Best	QD-Score	Cov	Best	QD-Score	Cov	Best
PPGA	$\mathbf{1.02 \times 10^5}$	**0.52**	**8324**	1.06×10^5	0.39	**4702**	$\mathbf{7.26 \times 10^5}$	0.58	**8919**	1.53×10^7	0.34	**7328**
PGA-ME	1.08×10^4	0.39	446	6.53×10^4	0.38	1621	3.31×10^5	0.31	4644	$\mathbf{1.79 \times 10^7}$	**0.37**	4571
sep-CMA-MAE	1.27×10^4	0.42	498	$\mathbf{1.84 \times 10^5}$	**0.64**	2326	6.03×10^5	**0.61**	228	1.16×10^7	0.30	1629
QDPG	6.53×10^3	0.26	412	6.59×10^4	0.35	1490	2.53×10^5	0.28	359	3.03×10^6	0.25	78
CMA-MAEGA (TD3, ES)	4.10×10^3	0.21	352	3.48×10^4	0.33	1025	5.70×10^5	0.56	2438	3.08×10^6	0.12	1020

Figure 4.6: Corrected CCDFs and Corrected QD metrics: QD-Score, *Cov*erage, *Best* Reward. Results are averaged over four seeds with error bars showing a 95% bootstrapped confidence interval.

Humanoid and Ant, and Coverage on Humanoid. The CCDF plots of the Corrected Archives show PPGA producing better distributions of policies on all tasks but Ant, suggesting PPGA's policies are robust to stochasticity.

4.5 Discussion and Limitations

We present a new method, PPGA, which is one of the first QD-RL methods to leverage on-policy RL, the first to solve the challenging Humanoid task, and the first to achieve equivalent performance in best reward compared to standard RL on all domains. We show that DQD algorithms and on-policy RL have emergent synergies that make them work particularly well with each other. However, instead of simply combining DQD and on-policy RL as is, we re-examine the fundamental assumptions and mechanisms of each component and implement changes that maximize their synergies.

There are some caveats with this approach. On-policy RL algorithms such as PPO are quite sample-inefficient and require many parallel environments per agent in order to compute the stochastic policy gradient. Although GPU acceleration and massive parallelism improve wall-clock convergence over off-policy RL, this makes our approach less sample-efficient than other off-policy QD-RL methods. Secondly, enabling PPO's adaptive standard deviation parameter (which is true by default for PPO) can have detrimental effects on PPGA's exploration capabilities, as made evident by the coverage score on Ant. This is mainly due to the fact that PPO favors collapsing the standard deviation to achieve higher average returns. In the future, we will investigate modifying the standard deviation parameter such that it dynamically shrinks or increases the standard deviation value based on the QD-optimization landscape as opposed to the RL one. Finally, we are interested to see how this method scales to even more data-rich regimes such as distributed settings, as well as its application to harder problems such as real robotics tasks. We leave these as potential avenues of future research.

Chapter 5

Quality Diversity for Long Evaluation Times

5.1 Introduction

* We present an efficient method of automatically generating a collection of environments that elicit diverse agent behaviors. As a motivating example, consider deploying a robot agent at scale in a variety of home environments. The robot should generalize by performing robustly not only in test homes, but in any end user's home. To validate agent generalization, the test environments should have good coverage for the robot agent. However, obtaining such coverage may be difficult, as the generated environments would depend on the application domain, e.g. kitchen or living room, and on the specific agent we want to test, since different agents exhibit different behaviors.

To enable generalization of autonomous agents to new environments with differing levels of complexity, previous work on open-ended learning [256, 257] has integrated the environment generation and the agent training processes. The interplay between the two processes acts as a natural curriculum for the agents to learn robust skills that generalize to new, unseen environments [55, 181, 56]. The performance of these agents has been evaluated either in environments from the training distribution [256, 257, 56] or in suites of manually authored environments [55, 122, 181].

As a step towards testing generalizable agents, there has been increasing interest in competitions [148, 106] that require agents to generalize to new game layouts. Despite the recent progress of deep learning agents in fixed game domains, e.g. in Chess [223], Go [222], Starcraft [252], and Poker [163, 29], it has been rule-based agents that have succeeded in these competitions [106]. Such competitions also rely on manually authored game levels as a test set, handcrafted by a human designer.

*Work co-led by Varun Bhatt and Matthew C. Fontaine in the ICAROS Lab.

While manually authored environments are important for standardized testing, creating these environments can be tedious and time-consuming. Additionally, manually authored test suites are often insufficient for eliciting the diverse range of possible agent behaviors. Instead, we would like an interactive test set that proposes an environment, observes the agent's performance and behavior, and then proposes new environments that diversify the agent behaviors, based on what the system has learned from previous execution traces of the agent.

To address collecting environments with diverse agent behaviors, prior work frames the problem as a quality diversity (QD) problem [74, 75, 71]. A QD problem consists of an objective function, e.g. whether the agent can solve the environment, and measure functions, e.g. how long the agent takes to complete their task. The measure functions quantify the behavior we would like to vary in the agent, allowing practitioners to specify the case coverage they would like to see in the domain they are testing. While QD algorithms can generate diverse collections of environments, they require a large number of environment evaluations to produce the collection, and each of these evaluations requires multiple time-consuming simulated executions of potentially stochastic agent policies.

We study how *deep surrogate models that predict agent performance can accelerate the generation of environments that are diverse in agent behaviors*. We draw upon insights from model-based quality diversity algorithms that have been previously shown to improve sample efficiency in design optimization [83] and Hearthstone deckbuilding [267]. Environments present a much more complex prediction task because the evaluation of environments involves simulating stochastic agent policies, and small changes in the environment may result in large changes in the emergent agent behaviors [231].

We make the following contributions: (1) We propose the use of deep surrogate models to predict agent performance in new environments. Our algorithm, Deep Surrogate Assisted Generation of Environments (DSAGE) (Fig. 5.1), integrates deep surrogate models into quality diversity optimization to efficiently generate diverse environments. (2) We show in two benchmark domains from previous work, a Maze domain [55, 181] with a trained ACCEL agent [181] and a Mario domain [125, 75] with an A* agent [15], that DSAGE outperforms state-of-the-art QD algorithms in discovering diverse agent behaviors. (3) We show with ablation studies that training the surrogate model with ancillary agent behavior data and downsampling a subset of solutions from the surrogate archive results in substantial improvements in performance, compared to the surrogate models of previous work [267].

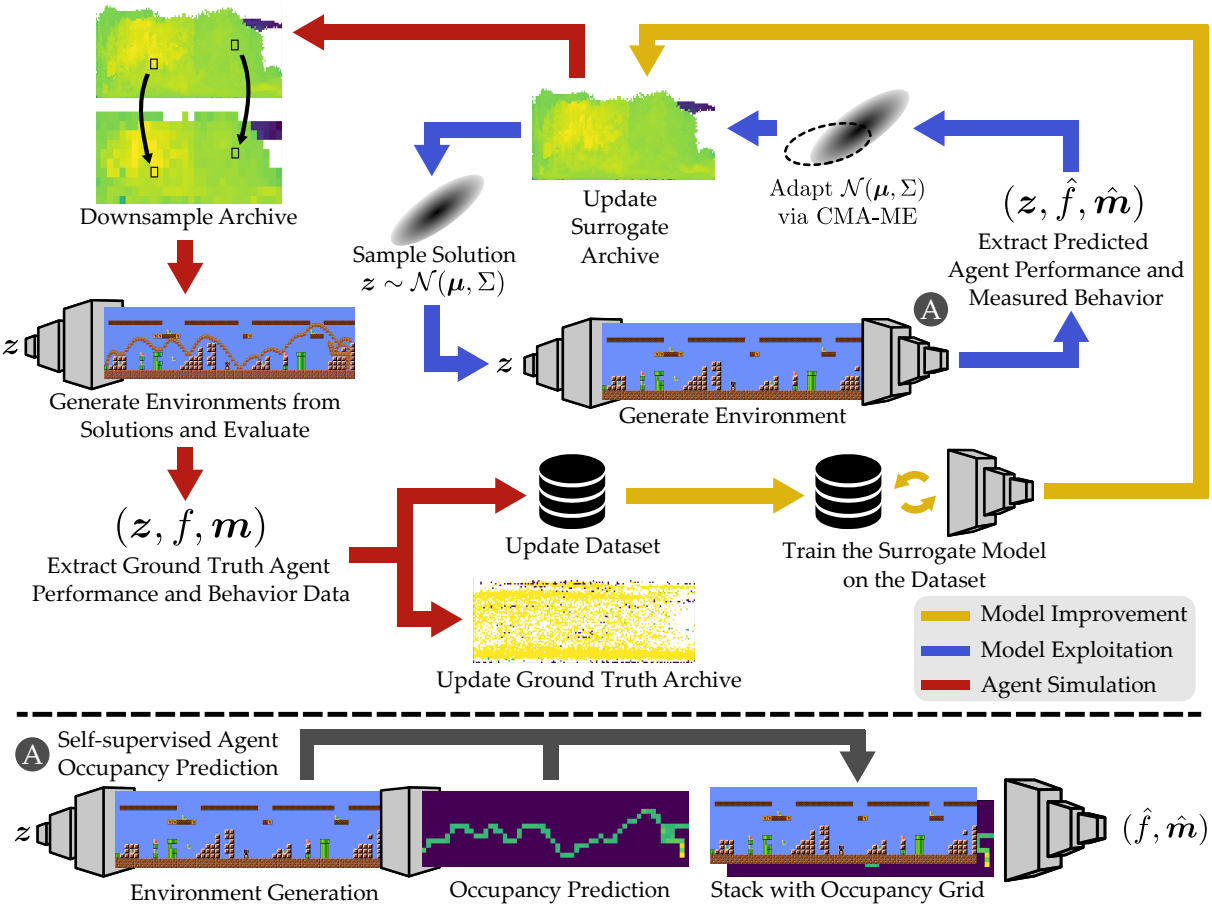


Figure 5.1: An overview of the Deep Surrogate Assisted Generation of Environments (DSAGE) algorithm. The algorithm begins by generating and evaluating random environments to initialize the dataset and the surrogate model (not shown in the figure). An archive of solutions is generated by exploiting a deep surrogate model (**blue arrows**) with a QD optimizer, e.g., CMA-ME [77]. A subset of solutions from this archive are chosen by downsampling and evaluated by generating the corresponding environment and simulating an agent (**red arrows**). The surrogate model is then trained on the data from the simulations (**yellow arrows**). While the images show Mario levels, the algorithm structure is similar for mazes.

5.2 Problem Definition

Quality diversity (QD) optimization. We adopt the QD problem definition from previous work [73]. A QD optimization problem specifies an objective function $f : \mathbb{R}^n \to \mathbb{R}$ and a joint measure function $\boldsymbol{m} : \mathbb{R}^n \to \mathbb{R}^m$. For each element $s \in S$, where $S \subseteq \mathbb{R}^m$ is the range of the measure function, the QD goal is to find a solution $\boldsymbol{\theta} \in \mathbb{R}^n$ such that $\boldsymbol{m}(\boldsymbol{\theta}) = s$ and $f(\boldsymbol{\theta})$ is maximized.

Since the range of the measure function can be continuous, we restrict ourselves to algorithms from the MAP-Elites family [49, 169] that discretize this space into a finite number of M cells. A solution $\boldsymbol{\theta}$ is mapped to a cell based

on its measure $\boldsymbol{m}(\boldsymbol{\theta})$. The solutions that occupy cells form an *archive* of solutions. Our goal is to find solutions $\boldsymbol{\theta}_i, i \in \{1, ..., M\}$ that maximize the objective f for all cells in the measure space.

$$\max_{\boldsymbol{\theta}_i} \sum_{i=1}^{M} f(\boldsymbol{\theta}_i) \tag{5.1}$$

The computed sum in Eq. 5.1 is defined as the QD-score [190], where empty cells have an objective value of 0. A second metric of the performance of a QD algorithm is coverage of the measure space, defined as the proportion of cells that are filled in by solutions: $\frac{1}{M} \sum_{i=1}^{M} \mathbf{1}_{\boldsymbol{\theta}_i}$.

QD for environment generation. We assume a single agent acting in an environment parameterized by $\boldsymbol{\theta} \in \mathbb{R}^n$. The environment parameters can be locations of different objects or latent variables that are passed as inputs to a generative model [92].[†] A QD algorithm generates new solutions $\boldsymbol{\theta}$ and evaluates them by simulating the agent on the environment parameterized by $\boldsymbol{\theta}$. The evaluation returns an objective value f and measure values $\boldsymbol{m}$. The QD algorithm attempts to generate environments that maximize f but are diverse with respect to the measures $\boldsymbol{m}$.

5.3 Background and Related Work

Quality diversity (QD) optimization. QD optimization originated in the genetic algorithm community with diversity optimization [145], the predecessor to QD. Later work introduced objectives to diversity optimization and resulted in the first QD algorithms: Novelty Search with Local Competition [146] and MAP-Elites [169, 49]. The QD community has grown beyond its genetic algorithm roots, with algorithms being proposed based on gradient ascent [73], Bayesian optimization [130], differential evolution [38], and evolution strategies [77, 44, 42]. QD algorithms have applications in damage recovery in robotics [49], reinforcement learning [239, 175, 186], and generative design [83, 104].

Among the QD algorithms, those of particular interest to us are the model-based ones. Current model-based [12, 161] QD algorithms either (1) learn a surrogate model of the objective and measure functions [83, 105, 32], e.g. a Gaussian process or neural network, (2) learn a generative model of the representation parameters [84, 195], or (3) draw inspiration from model-based RL [129, 152]. In particular, Deep Surrogate Assisted MAP-Elites (DSA-ME) [267] trains a deep surrogate model on a diverse dataset of solutions generated by MAP-Elites and then leverages the model

[†]For consistency with the generative model literature, we use $\mathbf{z}$ instead of $\boldsymbol{\theta}$ when denoting latent vectors

to guide MAP-Elites. However, DSA-ME has only been applied to Hearthstone deck building, a simpler prediction problem than predicting agent behavior in generated environments. Additionally, DSA-ME is specific to MAP-Elites only and cannot run other QD algorithms to exploit the surrogate model. Furthermore, DSA-ME is restricted to direct search and cannot integrate generative models to generate environments that match a provided dataset.

Automatic environment generation. Automatic environment generation algorithms have been proposed in a variety of fields. Methods between multiple communities often share generation techniques, but differ in how each community applies the generation algorithms.

For example, in the procedural content generation (PCG) field [217], an environment generator produces video game levels that result in player enjoyment. Since diversity of player experience and game mechanics is valued in games, many level generation systems incorporate QD optimization [95, 75, 61, 135, 230, 213, 208]. The procedural content generation via machine learning (PCGML) [233, 153] subfield studies environment generators that incorporate machine learning techniques such as Markov Chains [226], probabilistic graphical models [101], LSTMs [232], generative models [253, 89, 245, 209], and reinforcement learning [134, 60]. Prior work [126] has leveraged surrogate models trained on offline data to accelerate search-based PCG [244].

Environment generation methods have also been proposed by the scenario generation community in robotics. Early work explored automated methods for generating road layouts, vehicle arrangements, and vehicle behaviors for testing autonomous vehicles [10, 172, 2, 197, 87, 203, 80]. Outside of autonomous vehicles, prior work [269] evaluates robot motion planning algorithms by generating environments that target specific motion planning behaviors. In human-robot interaction, QD algorithms have been applied as environment generators to find failures in shared autonomy systems [71] and human-aware planners tested in the collaborative Overcooked domain [74]

Environment generation can also help improve the generality of RL agents. Prior work proposes directly applying PCG level generation algorithms to improve the robustness of RL [196, 124] or to benchmark RL agents [41]. Paired Open-ended Trailblazer (POET) [256, 257] coevolves a population of both agents and environments to discover specialized agents that solve complex tasks. POET inspired a variety of open-ended coevolution algorithms [82, 23, 57, 56]. Later work proposes the PAIRED [55], PLR [123, 122], and ACCEL [181] algorithms that train a single generally capable agent by maximizing the regret between a pair of agents. These methods generate environments in parallel with an agent to create an automatic training curriculum. However, the authors validate these methods on human-designed

environments [137]. Our work proposes a method that automatically generates valid environments that reveal diverse behaviors of these more general RL agents.

5.4 Deep Surrogate Assisted Generation of Environments (DSAGE)

Algorithm. We propose the Deep Surrogate Assisted Generation of Environments (DSAGE) algorithm for discovering environments that elicit diverse agent behaviors. Akin to the MAP-Elites family of QD algorithms, DSAGE maintains a *ground-truth archive* where solutions are stored based on their ground-truth evaluations. Simultaneously, DSAGE also trains and exploits a deep surrogate model for predicting the behavior of a fixed agent in new environments. The QD optimization occurs in three phases that take place in an outer loop: model exploitation, agent simulation, and model improvement (Fig. 5.1). Algorithm 9 provides the pseudocode for the DSAGE algorithm.

The model exploitation phase (lines 10–17) is an inner loop that leverages existing QD optimization algorithms and the predictions of the deep surrogate model to build an archive – referred to as the *surrogate* archive – of solutions. The first step of this phase is to query a list of B candidate solutions through the QD algorithm's *ask* method. These solutions are environment parameters, e.g., latent vectors of a GAN, which are passed through the environment generator, e.g., a GAN, to create an environment (line 14). Next, we make predictions with the surrogate model. The surrogate model first predicts data representing the agent's behavior, e.g., the probability of occupying each discretized tile in the environment (line 15), referred to as "ancillary agent behavior data" (y). The predicted ancillary agent behavior data ($\hat{y}$) then guides the surrogate model's downstream prediction of the objective ($\hat{f}$) and the measure values ($\hat{\boldsymbol{m}}$) (line 16). Finally, the QD algorithm's *tell* method adds the solution to the surrogate archive based on the predicted objective and measure values.

Note that since DSAGE is independent of the QD algorithm, the *ask* and *tell* methods abstract out the QD algorithm's details. For example, when the QD algorithm is MAP-Elites or CMA-ME, *tell* adds solutions if the cell in the measure space that they belong to is empty or if the existing solution in that cell has a lower objective. For CMA-ME, *tell* also includes updating internal CMA-ES parameters.

The agent simulation phase (lines 18–24) inserts a subset of solutions from the surrogate archive into the ground-truth archive. This phase begins by selecting the subset of solutions from the surrogate archive (line 18). The selected solutions are evaluated by generating the corresponding environment (line 20) and simulating a fixed agent to obtain the true objective and measure values, as well as ancillary agent behavior data (line 21). Evaluation data is appended to

Algorithm 9: Deep Surrogate Assisted Generation of Environments (DSAGE)

Input: N: Maximum number of evaluations, n_{rand}: Number of initial random solutions, $N_{exploit}$: Number of iterations in the model exploitation phase, B: Batch size for the model exploitation QD optimizer

Output: Final version of the ground-truth archive $\mathcal{A}_{gt}$

1 Initialize the ground-truth archive $\mathcal{A}_{gt}$, the dataset $\mathcal{D}$, and the deep surrogate model sm
2 $\boldsymbol{\Theta} \leftarrow$ *generate_random_solutions*(n_{rand})
3 **for** $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ **do**
4 $\quad env \leftarrow g(\boldsymbol{\theta})$
5 $\quad f, \boldsymbol{m}, \boldsymbol{y} \leftarrow$ *evaluate*(env)
6 $\quad \mathcal{D} \leftarrow \mathcal{D} \cup (\boldsymbol{\theta}, f, \boldsymbol{m}, \boldsymbol{y})$
7 $\quad \mathcal{A}_{gt} \leftarrow$ *add_solution*$(\mathcal{A}_{gt}, (\boldsymbol{\theta}, f, \boldsymbol{m}))$
8 $evals \leftarrow n_{rand}$
9 **while** $evals < N$ **do**
10 $\quad$ Initialize a QD optimizer qd with the surrogate archive $\mathcal{A}_{surrogate}$
11 $\quad$ **for** $itr \in \{1, 2, \ldots, N_{exploit}\}$ **do**
12 $\quad\quad \boldsymbol{\Theta} \leftarrow$ *qd.ask*(B)
13 $\quad\quad$ **for** $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ **do**
14 $\quad\quad\quad env \leftarrow g(\boldsymbol{\theta})$
15 $\quad\quad\quad \hat{y} \leftarrow$ *sm.predict_ancillary*(env)
16 $\quad\quad\quad \hat{f}, \hat{\boldsymbol{m}} \leftarrow$ *sm.predict*$(env, \hat{y})$
17 $\quad\quad\quad$ *qd.tell*$(\boldsymbol{\theta}, \hat{f}, \hat{\boldsymbol{m}})$
18 $\quad \boldsymbol{\Theta} \leftarrow$ *select_solutions*$(\mathcal{A}_{surrogate})$
19 $\quad$ **for** $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ **do**
20 $\quad\quad env \leftarrow g(\boldsymbol{\theta})$
21 $\quad\quad f, \boldsymbol{m}, \boldsymbol{y} \leftarrow$ *evaluate*(env)
22 $\quad\quad \mathcal{D} \leftarrow \mathcal{D} \cup (\boldsymbol{\theta}, f, \boldsymbol{m}, \boldsymbol{y})$
23 $\quad\quad \mathcal{A}_{gt} \leftarrow$ *add_solution*$(\mathcal{A}_{gt}, (\boldsymbol{\theta}, f, \boldsymbol{m}))$
24 $\quad\quad evals \leftarrow evals + 1$
25 $\quad sm.train(\mathcal{D})$

Model Exploitation

Agent Simulation

Model Improvement

the dataset, and solutions that improve their corresponding cell in the ground-truth archive are added to that archive (lines 22, 23).

In the model improvement phase (line 25), the surrogate model is trained in a self-supervised manner through the supervision provided by the agent simulations and the ancillary agent behavior data.

The algorithm is initialized by generating random solutions and simulating the agent in the corresponding environments (lines 2-7). Subsequently, every outer iteration (lines 9-25) consists of model exploitation followed by agent simulation and ending with model improvement.

Self-supervised prediction of ancillary agent behavior data. By default, a surrogate model directly predicts the objective and measure values based on the initial state of the environment and the agent (provided in the form of a one-hot encoded image). However, we anticipate that direct prediction will be challenging in some domains, as

it requires understanding the agent's trajectory in the environment. Thus, we provide additional supervision to the surrogate model in DSAGE via a two-stage self-supervised process.

First, a deep neural network predicts ancillary agent behavior data. In our work, we obtain this data by recording the expected number of times the agent visits each discretized tile in the environment, resulting in an "occupancy grid." We then concatenate the predicted ancillary information, i.e., the predicted occupancy grid, with the one-hot encoded image of the environment and pass them through another deep neural network to obtain the predicted objective and measure values. We use CNNs for both predictors. As a baseline, we compare our model with a CNN that directly predicts the objective and measure values without the help of ancillary data.

Downsampling to select solutions from the surrogate archive. After the model exploitation phase, the surrogate archive is populated with solutions that were predicted to be high-performing and diverse. Hence, a basic selection mechanism (line 18) would select all solutions from the surrogate archive, identical to DSA-ME [267]. However, if the surrogate archive is overly populated, full selection may result in a large number of ground-truth evaluations per outer-loop iteration, leading to fewer outer loops and less surrogate model training. To balance the trade-off between evaluating solutions from the surrogate archive and training the surrogate model, we only select a subset of solutions for evaluation by downsampling the surrogate archive. Downsampling uniformly divides the surrogate archive into sub-regions of cells and selects a random solution from each area.

5.5 Domains

We test our algorithms in two benchmark domains from prior work: a Maze domain [37, 55, 181] with a trained ACCEL agent [181] and a Mario domain [243, 75] with an A* agent [15]. We select these domains because, despite their relative simplicity (each environment is represented as a 2D grid of tiles), agents in these environments exhibit complex and diverse behaviors.

In the Maze domain, we directly search for different mazes, with the QD algorithm returning the layout of the maze. In the Mario domain, we search for latent codes that are passed through a pre-trained GAN, similar to the corresponding previous work.

We select the objective and measure functions as described below. Since the agent or the environment dynamics are stochastic in each domain, we average the objective and measure values over 50 episodes in the Maze domain and 5 episodes in the Mario domain.

Maze. We set a binary objective function f that is 1 if the generated environment is solvable and 0 otherwise, indicating the validity of the environment. Since we wish to generate visually diverse levels that offer a range of difficulty level for the agent, we select as measures (1) *number of wall cells* (range: $[0, 256]$), and (2) *mean agent path length* (range: $[0, 648]$, where 648 indicates a failure to reach the goal).

Mario. Since we wish to generate playable levels, we set the objective as the *completion rate*, i.e., the proportion of the level that the agent completes before dying. We additionally want to generate environments that result in qualitatively different agent behaviors, thus we selected as measures: (1) *sky tiles*, the number of tiles of a certain type that are in the top half of the 2D grid (range: $[0, 150]$), (2) *number of jumps*, the number of times that the A* agent jumps during its execution (range: $[0, 100]$).

5.6 Experiments

5.6.1 Experiment Design

Independent variables. In each domain (Maze and Mario), we follow a between-groups design, where the independent variable is the algorithm. We test the following algorithms:

DSAGE: The proposed algorithm that includes predicting ancillary agent behavior data and downsampling the surrogate archive (Sec. 5.4).

DSAGE-Only Anc: The proposed algorithm with ancillary data prediction and no downsampling, i.e., selecting all solutions from the surrogate archive.

DSAGE-Only Down: The proposed algorithm with downsampling and no ancillary data prediction.

DSAGE Basic: The basic version of the proposed algorithm that selects all solutions from the surrogate archive and does not predict ancillary data.

Baseline QD: The QD algorithm without surrogate assistance. We follow previous work [75] and use CMA-ME for the Mario domain. Since CMA-ME operates only in continuous spaces, we use MAP-Elites in the discrete Maze domain.

Domain Randomization (DR) [121, 202, 241]: Algorithm that generates and evaluates random solutions, i.e., wall locations in the maze domain and the latent code to pass through the GAN in the Mario domain.

Dependent variables. We measure the quality and diversity of the solutions with the QD-score metric [190](Eq. 5.1). As an additional metric of diversity, we also report the archive coverage. We run each algorithm for 5 trials in each domain.

Hypothesis. *We hypothesize that DSAGE will result in a better QD-score than DSAGE Basic in all domains, which in turn will result in better performance than the baseline QD algorithm. DSAGE, DSAGE Basic, and the baseline QD algorithm will all exceed DR.* We base this hypothesis on previous work [169, 71] which shows that QD algorithms outperform random sampling in a variety of domains, as well as previous work [83, 267] which shows that surrogate-assisted MAP-Elites outperforms standard MAP-Elites in design optimization and Hearthstone domains. Furthermore, we expect that the additional supervision through ancillary agent behavior data and downsampling will result in DSAGE performing significantly better than DSAGE Basic.

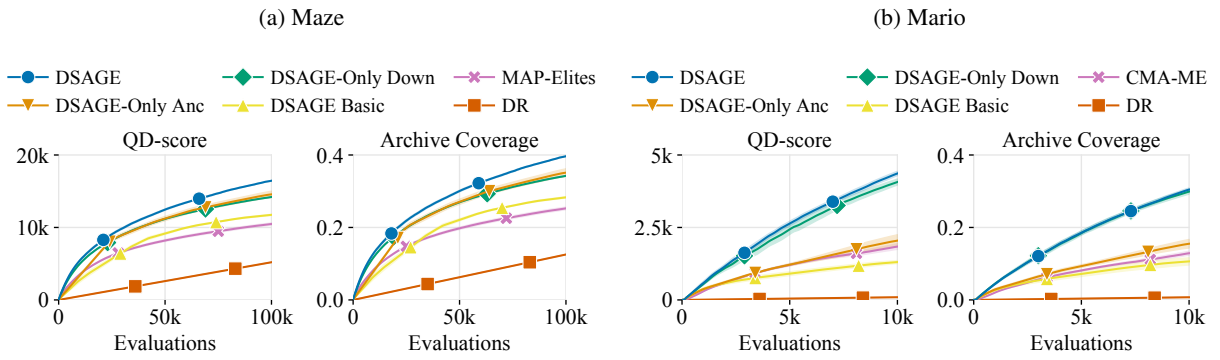
5.6.2 Analysis

Fig. 5.2 summarizes the results obtained by the six algorithms on the Maze and the Mario domains.

One-way ANOVA tests showed a significant effect of the algorithm on the QD-score for the Maze ($F(5, 24) = 430.98, p < 0.001$) and Mario ($F(5, 24) = 238.09, p < 0.001$) domains.

Post-hoc pairwise comparisons with Bonferroni corrections showed that DSAGE outperformed DSAGE Basic, Baseline QD, and DR in both the Maze and the Mario domains ($p < 0.001$). Additionally, DSAGE Basic outperformed MAP-Elites and DR in the Maze domain ($p < 0.001$), while it performed significantly worse than the QD counterpart, CMA-ME, in the Mario domain ($p = 0.003$). Finally, Baseline QD outperformed DR in both the Maze and Mario domains ($p < 0.001$).

These results show that deep surrogate assisted generation of environments results in significant improvements compared to quality diversity algorithms without surrogate assistance. They also show that adding ancillary agent



Algorithm	QD-score	Archive Coverage
DSAGE	**16,446.60 ± 42.27**	**0.40 ± 0.00**
DSAGE-Only Anc	14,568.00 ± 434.56	0.35 ± 0.01
DSAGE-Only Down	14,205.20 ± 40.86	0.34 ± 0.00
DSAGE Basic	11,740.00 ± 84.13	0.28 ± 0.00
MAP-Elites	10,480.80 ± 150.13	0.25 ± 0.00
DR	5,199.60 ± 30.32	0.13 ± 0.00

Algorithm	QD-score	Archive Coverage
DSAGE	**4,362.29 ± 72.54**	**0.30 ± 0.00**
DSAGE-Only Anc	2,045.28 ± 201.64	0.16 ± 0.01
DSAGE-Only Down	4,067.42 ± 102.06	**0.30 ± 0.01**
DSAGE Basic	1,306.11 ± 50.90	0.11 ± 0.01
CMA-ME	1,840.17 ± 95.76	0.13 ± 0.01
DR	92.75 ± 3.01	0.01 ± 0.00

Figure 5.2: QD-score and archive coverage attained by baseline QD algorithms and DSAGE in the Maze and Mario domains over 5 trials. Tables and plots show mean and standard error of the mean.

behavior data and downsampling is important in both domains. Without these components, DSAGE Basic has limited or no improvement compared to the QD algorithm without surrogate assistance. Additionally, domain randomization is significantly worse than DSAGE as well as the baselines. The archive coverage and consequently the QD-score is negligible in the Mario domain since randomly sampled latent codes led to little diversity in the levels.

Table 5.1 shows another metric of the speed-up provided by DSAGE: the number of evaluations (agent simulations) required to reach a fixed QD-score. We set this fixed QD-score to be 10480.8 in the Maze domain and 1306.11 in the Mario domain, which are the mean QD-scores of MAP-Elites and DSAGE Basic, respectively, in those domains. DSAGE reaches these QD-scores faster than the baselines do.

To assess the quality of the trained surrogate model, we create a combined dataset consisting of data from one run of each surrogate assisted algorithm. We use this dataset to evaluate the surrogate models trained from separate runs of DSAGE and its variants. Table 5.2 shows the mean absolute error (MAE) of the predictions by the surrogate models. The model learned by DSAGE Basic fails to predict the agent-based measures well. It has an MAE of 157.69 for the mean agent path length in Maze and MAE = 10.71 for the number of jumps in Mario. In contrast, the model learned by DSAGE makes more accurate predictions, with MAE = 96.58 for mean agent path length and MAE = 7.16 for number of jumps.

Table 5.1: Number of evaluations required to reach a QD-score of 10480.8 in the Maze domain and 1306.11 in the Mario domain.

(a) Maze

Algorithm	Evaluations
DSAGE	**33,930.40 ± 1,411.04**
DSAGE-Only Anc	51,919.60 ± 8,254.24
DSAGE-Only Down	42,816.60 ± 691.38
DSAGE Basic	85,328.60 ± 2,947.24
MAP-Elites	100,000

(b) Mario

Algorithm	Evaluations
DSAGE	**2,464.40 ± 356.36**
DSAGE-Only Anc	7,727.40 ± 1,433.33
DSAGE-Only Down	**2,768.60 ± 586.34**
DSAGE Basic	10,000
CMA-ME	5,760.00 ± 516.14

Table 5.2: Mean absolute error of the objective and measure predictions by the surrogate models.

	Maze			Mario		
Algorithm	Objective MAE	Number of Wall Cells MAE	Mean Agent Path Length MAE	Objective MAE	Number of Sky Tiles MAE	Number of Jumps MAE
DSAGE	0.03	0.37	96.58	0.10	1.10	7.16
DSAGE-Only Anc	0.04	0.96	95.14	0.20	1.11	9.97
DSAGE-Only Down	0.10	0.95	151.50	0.11	0.87	6.52
DSAGE Basic	0.18	5.48	157.69	0.20	2.16	10.71

5.6.3 Ablation Study

Sec. 5.4 describes two key components of DSAGE: (1) self-supervised prediction of ancillary agent behavior data, and (2) downsampling to select solutions from the surrogate archive. We perform an ablation study by treating the inclusion of ancillary data prediction (ancillary data / no ancillary data) and the method of selecting solutions from the surrogate archive (downsampling / full selection) as independent variables. A two-way ANOVA for each domain showed no significant interaction effects. We perform a main effects analysis for each independent variable.

Inclusion of ancillary data prediction. A main effects analysis for the inclusion of ancillary data prediction showed that algorithms that predict ancillary agent behavior data (DSAGE, DSAGE-Only Anc) performed significantly better than their counterparts with no ancillary data prediction (DSAGE-Only Down, DSAGE Basic) in both domains ($p < 0.001$).

Fig. 5.2 shows that predicting ancillary agent behavior data also resulted in a larger mean coverage for Maze, while it has little or no improvement for Mario. Additionally, as shown in Table 5.2, predicting ancillary agent behavior data helped improve the prediction of the mean agent path length in the Maze domain but provided little improvement to the prediction of the number of jumps in the Mario domain. The reason is that in the Maze domain, the mean agent path length is a scaled version of the sum of the agent's tile occupancy frequency, hence the two-stage process which predicts the occupancy grid first is essential for improving the accuracy of the model. On the other hand, the presence

of a jump in Mario depends not only on cell occupancy, but also on the structure of the level and the sequence of the occupied cells.

Method of selecting solutions from the surrogate archive. A main effects analysis for the method of selecting solutions from the surrogate archive showed that the algorithms with downsampling (DSAGE, DSAGE-Only Down) performed significantly better than their counterparts with no downsampling (DSAGE-Only Anc, DSAGE Basic) in both domains ($p < 0.001$).

A major advantage of downsampling is that it decreases the number of ground-truth evaluations in each outer iteration. Thus, for a fixed evaluation budget, downsampling results in a greater number of outer iterations. For instance, in the Maze domain, runs without downsampling had only 6-7 outer iterations, while runs with downsampling had approximately 220 outer iterations. More outer iterations leads to more training and thus higher accuracy of the surrogate model. In turn, a more accurate surrogate model will generate a better surrogate archive in the inner loop.

We ran an ablation to test between two possible explanations for why having more outer iterations helps with performance: (1) larger number of training epochs, (2) more updates to the dataset allowing the surrogate model to iteratively correct its own errors. We observed that iterative correction accounted for most of the performance increase with downsampling.

The second advantage of downsampling is that it selects solutions evenly from all regions of the measure space, thus creating a more balanced dataset. This helps train the surrogate model in parts of the measure space that are not frequently visited. We compared against an algorithm that selects a subset of solutions uniformly at random from the surrogate archive instead of downsampling. We observe that downsampling has a slight advantage over uniform random sampling in the Maze domain.

Furthermore, if instead of downsampling we sampled multiple solutions from nearby regions of the surrogate archive, the prediction errors could cause the solutions to collapse to a single cell in the ground-truth archive, resulting in many solutions being discarded.

Overall, our ablation study shows that both predicting the occupancy grid as ancillary data and downsampling the surrogate archive independently help improve the performance of DSAGE.

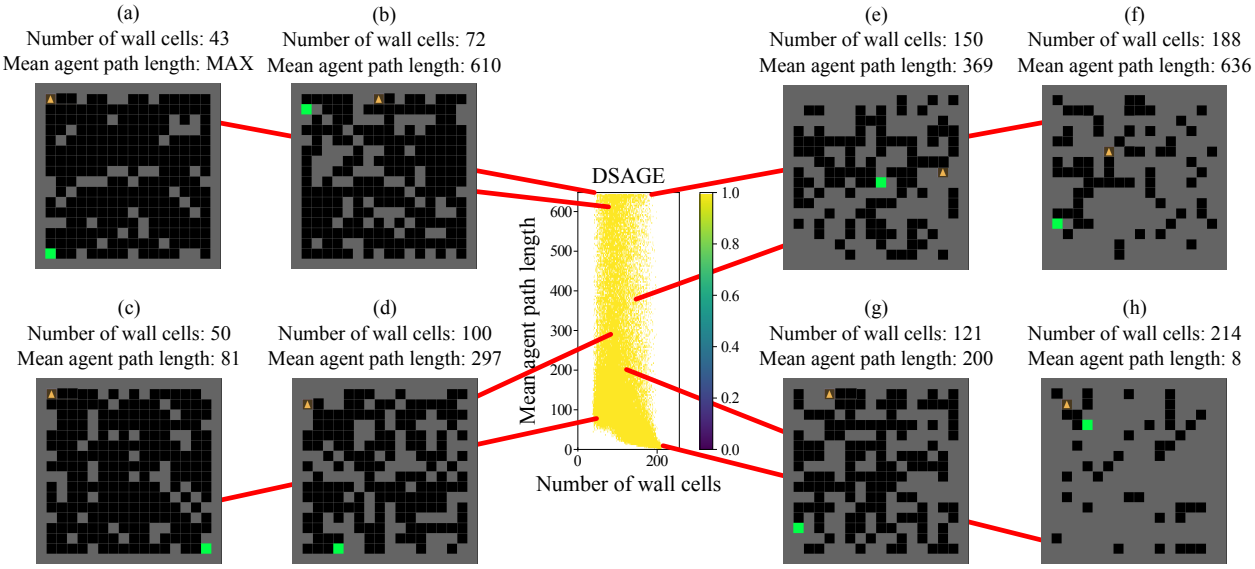


Figure 5.3: Archive and levels generated by DSAGE in the Maze domain. The agent's initial position is shown as an orange triangle, while the goal is a green square.

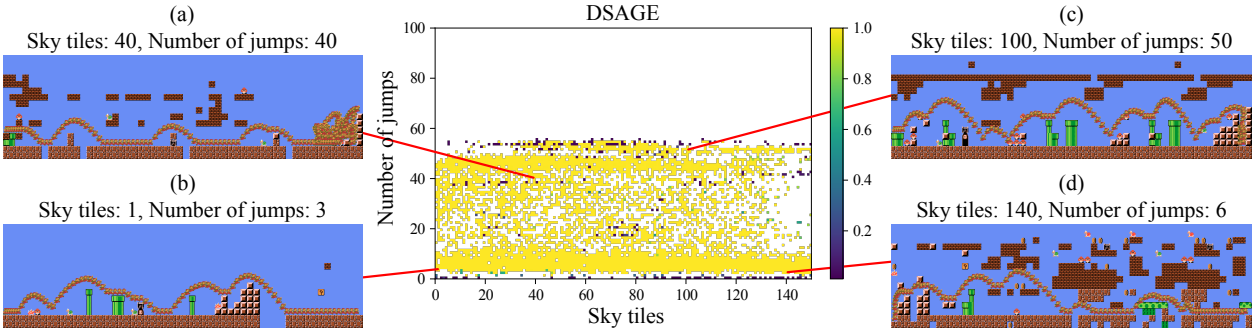


Figure 5.4: Archive and levels generated by DSAGE in the Mario domain. Each level shows the path Mario takes, starting on the left of the level and finishing on the right.

5.6.4 Qualitative Results

Fig. 5.3 and Fig. 5.4 show example environments generated by DSAGE in the Maze and Mario domains.

Having the mean agent path length as a measure in the Maze domain results in environments of varying difficulty for the ACCEL agent. For instance, we observe that the environment in Fig. 5.3(a) has very few walls, yet the ACCEL agent gets stuck in the top half of the maze and is unable to find the goal within the allotted time. On the other hand, the environment in Fig. 5.3(d) is cluttered and there are multiple dead-ends, yet the ACCEL agent is able to reach the goal.

Fig. 5.4 shows that the generated environments result in qualitatively diverse behaviors for the Mario agent too. Level (b) only has a few sky tiles and is mostly flat, resulting in a small number of jumps. Level (c) has a "staircase trap" on the right side, forcing the agent to perform continuous jumps to escape and complete the level.

5.7 Societal Impacts

By introducing surrogate models into quality diversity algorithms, we can efficiently generate environments that result in diverse agent behaviors. While we focused on an RL agent in a Maze domain and a symbolic agent in a Mario game domain, our method can be applied to a variety of agents and domains. This can help with testing the robustness of agents, attaining insights about their behavior, and discovering edge cases before real-world deployment. Furthermore, we anticipate that in the future, closing the loop between environment generation and agent training can improve the ability of agents to generalize to new settings and thus increase their widespread use.

Our work may also have negative impacts. Training agents in diverse environments can be considered as a step towards open-ended evolution [228], which raises concerns about the predictability and safety of the emergent agent behaviors [62, 115]. Discovering corner cases that result in unwanted behaviors or catastrophic failures may also be used maliciously to reveal vulnerabilities in deployed agents [200].

5.8 Limitations and Future Work

Automatic environment generation is a rapidly growing research area with a wide range of applications, including designing video game levels [217, 233, 153], training and testing autonomous agents [196, 41, 256, 55, 181], and discovering failures in human-robot interaction [71, 75]. We introduce the DSAGE algorithm, which efficiently generates a diverse collection of environments via deep surrogate models of agent behavior.

Our work has several limitations. First, occupancy grid prediction does not encode temporal information about the agent. While this prediction allows us to avoid the compounding error problem of model-based RL [262], forgoing temporal information makes it harder to predict some behaviors, such as the number of jumps in Mario. We will explore this trade-off in future work.

Furthermore, we have studied 2D domains where a single ground-truth evaluation lasts between a few seconds and a few minutes. We are excited about the use of surrogate models to predict the performance of agents in more complex domains with expensive, high-fidelity simulations [261].

Chapter 6

Quality Diversity for High-Dimensional Measures

6.1 Introduction

We present a method that enhances exploration in quality diversity (QD) optimization and show how this method enables new applications for QD. QD [190] is a branch of stochastic optimization that seeks to find diverse, high-performing solutions to a problem, with applications like robotics [169], generative modeling [58], and LLM red-teaming [206]. To illustrate, consider searching for "a photo of a hiker." By itself, this problem is ambiguous since hikers vary widely in appearance, depending on, for instance, where they are hiking and the time of year. If we run a single-objective optimization algorithm like Adam [136] or CMA-ES [108], the image we find would optimize the objective f of "a photo of a hiker," but the hiker could take on one of many different appearances. In contrast, QD [190] can manage the ambiguity by searching for an archive (set) of images that both optimize the objective f and diversify along the outputs of a measure function $\boldsymbol{m}$.

Prior work [190] typically hand-designs $\boldsymbol{m}$ to output low-dimensional ($<$10D) vectors. To illustrate, our measure function could output two measures: the hiker's age and the temperature for which they are dressed. Then, our archive would contain images like a younger hiker dressed for cold weather and an older hiker dressed for warm weather, as well as all images in between.

One reason prior works focus on low-dimensional measures is that high-dimensional measure spaces are prone to *distortion*, where many solutions (images) map to a small region of measure space (i.e., the solutions have similar measures). For example, it may be easy to search for images of hikers in warm weather, while hikers in cold weather

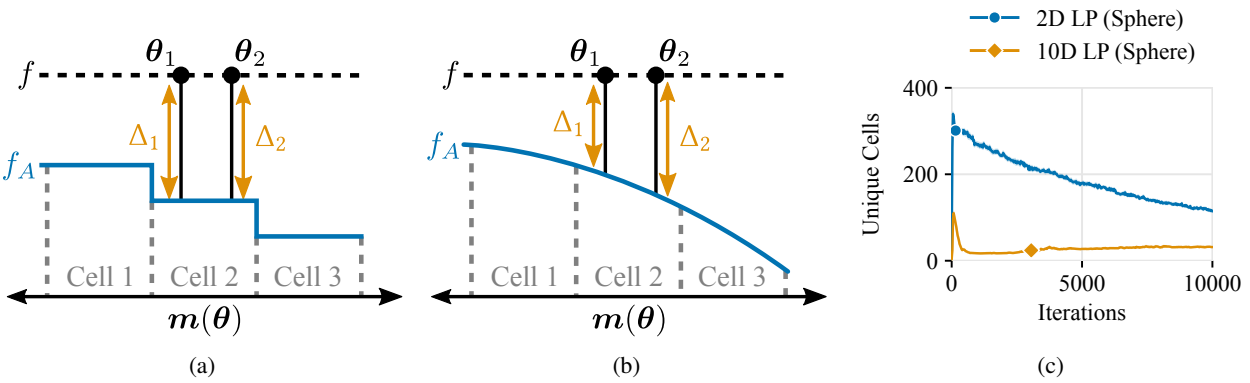


Figure 6.1: (a): One failure mode of CMA-MAE. On a flat objective f, solutions $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ fall in the same archive cell based on their measures, resulting in identical discount values from the discount function f_A. (b): In our proposed DMS, the discount model provides a smooth discount function that assigns distinct discount values to $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$, showing that $\boldsymbol{\theta}_2$ has greater archive improvement than $\boldsymbol{\theta}_1$ ($\Delta_2 > \Delta_1$) and thus providing a stronger signal to guide search. (c): Number of unique cells where solutions sampled by CMA-MAE land in two benchmarks (mean over 20 trials; Sec. 6.4).

are hard to find. Although distortion exists in low-dimensional measure spaces [73, 77], it is more prominent in high-dimensional measure spaces because there are exponentially larger volumes to which solutions can map (Sec. 6.4).

We propose to scale to high-dimensional measure spaces by addressing the effects of distortion in Covariance Matrix Adaptation MAP-Annealing (CMA-MAE) [72], a state-of-the-art black-box QD algorithm. To fill the archive of solutions, CMA-MAE searches for solutions $\boldsymbol{\theta}$ that maximize *archive improvement*, defined as $\Delta(\boldsymbol{\theta}) = f(\boldsymbol{\theta}) - f_A(\boldsymbol{m}(\theta))$, where $f(\boldsymbol{\theta})$ is the solution's objective value and f_A is a *discount function* that returns scalar *discount values* based on the solution's measures $\boldsymbol{m}(\boldsymbol{\theta})$. CMA-MAE represents f_A as a histogram by tessellating the measure space into cells and storing a scalar value in each cell. In domains with high distortion, particularly domains with high-dimensional measures, solutions with similar measures fall into the same cell. As a result, CMA-MAE incorrectly assigns these solutions the same discount value, which creates inaccurate improvement values that cause the search to stagnate (Fig. 6.1a).

Our key insight is that a smooth, continuous representation of the discount function will enhance exploration in high-dimensional measure spaces. As such, we propose Discount Model Search (DMS), where a smooth *discount model* accurately assigns distinct discount values to solutions even when distortion causes them to have similar measures (Fig. 6.1b). The discount values guide DMS to explore the measure space and discover solutions long after CMA-MAE would stagnate.

Figure 6.2: In the LSI (Hiker) domain, the objective is "A photo of the face of a hiker," and the measure space is the space of images. We specify desired measures with landscape images from LHQ [225]. Thus, DMS finds images depicting what a hiker might look like in each landscape: hikers in thick jackets for the mountains or lighter clothing for the beach, and even a baby bundled up for the snow. Each hiker is shown to the left of their corresponding landscape.

We show that by scaling to high-dimensional measure spaces, DMS facilitates new capabilities for QD. For example, it can be challenging to design a low-dimensional measure function to describe "where the hiker is located," as locations vary widely from beaches to mountains to forests. In general, creating measure functions can be tedious and unintuitive — similar to objective functions [224, 144], the design of the measure function vastly affects the quality of solutions produced [24, 189]. In contrast, consider that by defining the measures as age and temperature in our earlier example, we essentially specified a 2D grid of (age, temperature) points for which we sought images of hikers. If we treat the high-dimensional space of images (e.g., $256 \times 256 \times 3$ RGB vectors) as the measure space, we can replace the 2D points with images, i.e., we can easily specify "where the hiker is located" by providing a dataset of landscape images (Fig. 6.2). Thus, by scaling to high-dimensional measure spaces, we believe DMS makes QD more accessible: it enables specifying measures via datasets, which can alleviate the need for manual measure function design and enable specifying new types of measures. We refer to this setup as *Quality Diversity with Datasets of Measures (QDDM)*.

Our contributions are as follows: **(1)** We propose Discount Model Search (DMS), which improves exploration in high-dimensional measure spaces by searching over a smooth representation of the discount function (Sec. 6.5). **(2)** We benchmark DMS on standard QD domains (Sec. 6.6.1), showing it outperforms CMA-MAE and other black-box QD algorithms (Sec. 6.7). **(3)** We propose the QDDM setting and introduce two QDDM domains where images form the measure space (Sec. 6.6.2), showing that DMS also outperforms existing algorithms in these domains (Sec. 6.7). Overall, given the ubiquity of datasets in machine learning, we are excited for applications that can be framed as QDDM problems and tackled with DMS.

6.2 Problem Formulation

As formulated in prior work [73], black-box quality diversity (QD) optimization considers a scalar-valued *objective* function $f : \mathbb{R}^n \to \mathbb{R}$ and a vector-valued *measure* function $\boldsymbol{m} : \mathbb{R}^n \to \mathbb{R}^k$. Both functions take as input a solution $\boldsymbol{\theta} \in \mathbb{R}^n$, and $\boldsymbol{m}$ outputs k measures. The image of $\boldsymbol{m}$ forms the *measure space* S. The *QD objective* is to find, for every $\boldsymbol{s} \in S$, a solution $\boldsymbol{\theta}$ such that $\boldsymbol{m}(\boldsymbol{\theta}) = \boldsymbol{s}$ and $f(\boldsymbol{\theta})$ is maximized. As stated, this QD objective requires infinite memory since S is a continuous space. Hence, algorithms based on MAP-Elites [169] discretize S into a *tessellation* T of M *cells*, leading to a relaxed QD objective $\max_{\boldsymbol{\theta}_{1..M}} \sum_{i=1}^{M} f(\boldsymbol{\theta}_i)$. Each solution $\boldsymbol{\theta}_i$ has measures located in the region of measure space indicated by cell i in T, and the set of solutions $\boldsymbol{\theta}_{1..M}$ is referred to as an *archive* $\mathcal{A}$.

6.3 Background

Our work builds on several black-box QD algorithms.

MAP-Elites [169] produces a "grid archive" where the tessellation T divides the measure space into a grid of axis-aligned (hyper-)rectangles. Each cell in the archive stores one solution. Each iteration, MAP-Elites selects an archive solution $\boldsymbol{\theta}$, mutates it to create a new solution $\boldsymbol{\theta}'$, and adds $\boldsymbol{\theta}'$ to the archive. As $\boldsymbol{\theta}'$ is added, it is assigned to a cell e based on its measure values. $\boldsymbol{\theta}'$ replaces the solution in cell e if it has a higher objective value. In this manner, MAP-Elites retains *elites*, i.e., the best solution found in each cell. We consider a version of MAP-Elites that mutates solutions by adding isotropic Gaussian noise, i.e., new solutions are created as $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta} + \sigma\mathcal{N}(\mathbf{0}, \boldsymbol{I})$.

Since grid archives require exponentially more memory in high-dimensional measure spaces, prior work [248] proposes defining the tessellation T as a centroidal Voronoi tessellation (CVT), where a number of centroids (e.g., 10,000) divide the measure space into equally-sized Voronoi cells. We use these *CVT archives* in our experiments in high-dimensional measure spaces.

MAP-Elites (line) [249] augments MAP-Elites with the Iso+LineDD operator, which leverages correlations between solutions in the archive by generating new solutions as $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}_1 + \sigma_1\mathcal{N}(\mathbf{0}, \boldsymbol{I}) + \sigma_2\mathcal{N}(\mathbf{0}, \boldsymbol{I})(\boldsymbol{\theta}_2 - \boldsymbol{\theta}_1)$. $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ are sampled from the archive.

Covariance Matrix Adaptation MAP-Annealing (CMA-MAE) [72] is a state-of-the-art black-box QD algorithm that integrates the CMA-ES [108] optimizer into MAP-Elites to directly optimize the QD objective (Sec. 6.2). CMA-MAE

maintains a CMA-ES instance that samples solutions $\boldsymbol{\theta}_i$ from a Gaussian distribution $\mathcal{N}(\boldsymbol{\theta}^*, \boldsymbol{\Sigma})$. Each solution $\boldsymbol{\theta}_i$ is evaluated and added to the archive. For each $\boldsymbol{\theta}_i$, CMA-MAE computes an *improvement value* Δ_i that represents how much $\boldsymbol{\theta}_i$ improves the archive. The ranking of the improvement values enables the CMA-ES instance to update the distribution parameters $\boldsymbol{\theta}^*$ and $\boldsymbol{\Sigma}$ in the direction of greatest archive improvement; thus, CMA-ES continues to sample solutions that improve the archive in future iterations. This update causes CMA-ES to optimize the QD objective. Multiple CMA-ES instances may operate in parallel, with each instance referred to as an *emitter*.

The improvement value in CMA-MAE is defined as $\Delta(\boldsymbol{\theta}) = f(\boldsymbol{\theta}) - f_A(\boldsymbol{m}(\boldsymbol{\theta}))$, where $f_A : \mathbb{R}^k \to \mathbb{R}$ is a *discount function*. CMA-MAE represents f_A as a histogram in measure space by associating a *discount value* with each cell e. In CMA-MAE's predecessor, CMA-ME [77], the discount value is the objective value of the solution $\boldsymbol{\theta}_e$ currently in the cell, i.e., $f(\boldsymbol{\theta}_e)$. When a new solution is added to cell e, the discount value is updated to the objective of the new solution. As in MAP-Elites, a new solution can only replace the cell solution $\boldsymbol{\theta}_e$ if it has a higher objective value.

CMA-MAE builds on the insight that CMA-ME's discount values can cause the search to quickly leave areas of the archive that require further optimization of the objective [239]. For example, consider if the maximum objective attainable for a cell e is 100, and CMA-ME finds a solution with objective 90. Future solutions that land in e garner little improvement since the discount value associated with cell e is now $f(\boldsymbol{\theta}_e)$, which is 90. Even a solution $\boldsymbol{\theta}$ with objective $f(\boldsymbol{\theta}) = 100$ only receives an improvement of $100 - 90 = 10$. Thus, CMA-ME immediately searches for solutions in other areas of the archive that offer higher improvement values. In contrast, to continue optimizing for solutions that land in cell e, CMA-MAE sets the discount value to an *acceptance threshold* t_e, rather than the objective value of the solution in the cell. t_e is initialized to a minimum value f_{min}. As a new solution $\boldsymbol{\theta}'$ enters cell e, t_e is updated as $t_e \leftarrow (1-\alpha)t_e + \alpha f(\boldsymbol{\theta}')$, where $0 \leq \alpha \leq 1$ is an *archive learning rate* that controls how quickly t_e updates. Consider a cell e with $t_e = f_{min} = 0$. Given $\alpha = 0.1$, a solution with objective $f(\boldsymbol{\theta}') = 90$ updates t_e as $t_e \leftarrow (1 - 0.1) * 0 + 0.1 * 90 = 9$. A new solution in e with objective 100 would receive improvement $100 - 9 = 81$, so CMA-MAE still receives high improvement for discovering solutions in cell e.

Notably, when $\alpha = 1.0$, CMA-MAE focuses on exploration and is equivalent to CMA-ME, since t_e will always be set to the objective value of new solutions. When $\alpha = 0.0$, CMA-MAE performs single-objective optimization because t_e will be constant, so the improvement only considers the objective, i.e., $\Delta(\boldsymbol{\theta}) = f(\boldsymbol{\theta}) - C$. Thus, adjusting the learning rate α enables smoothly balancing between optimization of the objective and exploration of the measure space.

Density Descent Search (DDS) [142] removes the discount function from CMA-MAE and introduces a kernel density estimator (KDE) that models the density of previously discovered solutions in measure space. Instead of archive improvement, DDS ranks solutions by the KDE's density estimates, prioritizing solutions in areas of low density. The KDE provides a smoother signal than CMA-MAE's discrete histogram, enabling DDS to excel at exploring measure spaces. However, since the KDE does not consider the objective, DDS does not optimize the objective, making it a *diversity optimization* algorithm, i.e., it only searches for solutions with diverse measures. Nevertheless, we draw inspiration from how the smooth signal in DDS enhances exploration.

6.4 Understanding Distortion in High-Dimensional Measure Spaces

Distortion in QD refers to when large areas of solution space map to a small region of measure space [72]. When CMA-MAE encounters distortion, the solutions it samples land in fewer archive cells since they have similar measures. Fig. 6.1a shows one scenario where solutions that land in the same cell interfere with CMA-MAE's improvement mechanism. On a flat (constant) objective like the one in the figure, CMA-MAE's histogram represents how often it has visited each area of measure space, and the improvement ranking guides it towards areas that it has not visited before. For instance, Cell 3 has the lowest discount value among the three cells since it has not been explored yet, so the direction of greatest archive improvement is to generate solutions that land in Cell 3. However, since $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ both land in Cell 2, they have the same discount value. Since they also have the same constant objective, they receive the same improvement value ($\Delta_1 = \Delta_2$). Hence, CMA-MAE cannot identify the direction of greatest archive improvement.

In Fig. 6.1c, we present an experiment that illustrates how distortion causes solutions to land in the same cell, which activates failures like the one above. We run CMA-MAE in the 2D and 10D LP (Sphere) benchmarks, which are designed to exhibit distortion (Sec. 6.6.1). 2D and 10D indicate the measure space dimensionality. We plot the number of unique archive cells where solutions sampled by CMA-MAE land according to their measures; CMA-MAE samples 540 solutions per iteration. The plot shows that in both benchmarks, CMA-MAE begins searching in areas of low distortion, as many solutions land in unique cells. Over time, solutions more often land in the same cell (i.e., the number of unique cells goes down), indicating CMA-MAE has reached areas with higher distortion.

While low-dimensional measure spaces can exhibit distortion [142], our experiment shows how higher dimensions can amplify its effects. To elaborate, in Fig. 6.1c, the number of unique cells falls to only 30 in the higher-dimensional 10D LP (Sphere). In part, this occurs because although both the 2D and 10D benchmarks have archives with 10,000 cells, the cells in the 10D domain are exponentially larger by nature of being higher-dimensional. As such, there is a larger area of measure space where solutions sampled by CMA-MAE can fall and still be assigned the same discount value, leading to inaccurate improvement values that stall the search.

Since large cells can amplify the effects of distortion, prior work [72] suggests increasing the archive resolution (i.e., adding more cells), albeit only in a 2D measure space. With higher resolution, cells are smaller, so solutions with similar measures can still fall in different cells and receive different discount values. However, this approach entails large amounts of memory, and this amount grows exponentially with measure space dimensionality. Since increasing the archive resolution effectively makes the histogram closer to a continuous function, we propose to eliminate the histogram entirely and instead search with a continuous representation of the discount function.

6.5 Discount Model Search

To improve exploration in domains with distorted, high-dimensional measure spaces, we propose Discount Model Search (DMS). DMS trains a *discount model* to provide a smooth, continuous representation of the discount function. The key insight of DMS is that such a representation provides distinct discount values and hence improvement values, even when solutions have similar measures, making it easier to guide search towards solutions that improve the archive. For example, in Fig. 6.1b, the higher improvement Δ_2 correctly indicates that generating solutions in the direction of $\boldsymbol{\theta}_2$ would create greater archive improvement, as such solutions would land in Cell 3, which currently has a low discount value. Below we describe DMS's components, with pseudocode shown in Algorithm 10.

Archive and Emitters. DMS maintains a MAP-Elites-style archive that retains the best solutions found (line 14-16), and CMA-ES-based [108] emitters that optimize for archive improvement. Unlike CMA-MAE, DMS does not store discount or threshold values in the archive.

Discount Model. The primary component of DMS is its discount model, which approximates the true discount function $f_A : \mathbb{R}^k \to \mathbb{R}$. The discount model is a neural network $\hat{f}_A(\cdot; \psi)$ parameterized by ψ. It takes measures as input and outputs scalar discount values. While alternative models like kernel-based methods [36] are feasible, we select neural

Algorithm 10: Discount Model Search (DMS)

1 **Discount Model Search** $(eval, \boldsymbol{\theta}_0, N, W, \lambda, \sigma, n_{init}, \alpha, n_{empty}, f_{min})$**:**
  Input: $eval$ function that computes objective f and measures $\boldsymbol{m}$, initial solution $\boldsymbol{\theta}_0$, iterations N, num. emitters W, batch size λ, initial step size σ, initial training points n_{init}, archive learning rate α, num. empty points n_{empty}, minimum objective f_{min}
  Result: Generates $NW\lambda$ solutions, storing elites in an archive $\mathcal{A}$
2 Initialize empty archive $\mathcal{A}$; discount model $\hat{f}_A(\cdot;\psi)$ with parameters ψ
3 Sample n_{init} cells from $\mathcal{A}$ and regress $\hat{f}_A$ to output f_{min} at the centers of these cells
4 Initialize W emitters, each with mean $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}_0$, covariance $\boldsymbol{\Sigma} \leftarrow \sigma \boldsymbol{I}$, internal parameters $\boldsymbol{p}$
5 **for** $iter \leftarrow 1..N$ **do**
6 $\mathcal{D}_A \leftarrow []$ *// Dataset of measures and discount value targets.*
7 **for** Emitter 1 .. Emitter W **do**
8 **for** $i \leftarrow 1..\lambda$ **do**
9 $\boldsymbol{\theta}_i \sim \mathcal{N}(\boldsymbol{\theta}^*, \boldsymbol{\Sigma})$
10 $f(\boldsymbol{\theta}_i), \boldsymbol{m}(\boldsymbol{\theta}_i) \leftarrow eval(\boldsymbol{\theta}_i)$
11 $\Delta_i \leftarrow f(\boldsymbol{\theta}_i) - \hat{f}_A(\boldsymbol{m}(\boldsymbol{\theta}_i))$ *// Compute improvement based on discount model.*
12 Compute $t_{A,i}$ with Eq. 6.1, where $\boldsymbol{s} = \boldsymbol{m}(\boldsymbol{\theta}_i)$
13 $\mathcal{D}_A$.append$((\boldsymbol{m}(\boldsymbol{\theta}_i), t_{A,i}))$
14 $e \leftarrow$ calculate_archive_cell$(\mathcal{A}, \boldsymbol{m}(\boldsymbol{\theta}_i))$
15 **if** $f(\boldsymbol{\theta}_i) > f(\boldsymbol{\theta}_e)$ **then**
16 Replace $\boldsymbol{\theta}_e$ (the solution in cell e) with $\boldsymbol{\theta}_i$
17 Rank $\boldsymbol{\theta}_i$ by Δ_i
18 Adapt $\boldsymbol{\theta}^*, \boldsymbol{\Sigma}, \boldsymbol{p}$ based on improvement ranking Δ_i
19 **if** CMA-ES converges **then**
20 Restart emitter with $\boldsymbol{\theta}^* \leftarrow$ a random solution from $\mathcal{A}$, $\boldsymbol{\Sigma} \leftarrow \sigma \boldsymbol{I}$, new internal parameters $\boldsymbol{p}$
21 Sample n_{empty} unoccupied cells $e_{1..n_{empty}}$ from archive $\mathcal{A}$ without replacement
22 Compute centers $\boldsymbol{s}_{1..n_{empty}}$ of cells $e_{1..n_{empty}}$
23 $\mathcal{D}_A$.extend$((\boldsymbol{s}_{1..n_{empty}}, f_{min}))$
24 Regress $\hat{f}_A$ on $\mathcal{D}_A$

networks because the inductive biases of their various architectures make them suitable for many types of measures. For example, if the measures are low-dimensional vectors as is common in QD, an MLP (Multi-Layer Perceptron) would suffice. If the measure space includes high-dimensional data like images or text, a convolutional network [113] or transformer [250] would be suitable.

DMS trains the discount model as follows. First, to reflect that the archive is initially empty, DMS regresses the discount model to output the minimum objective f_{min} at the centers of n_{init} cells sampled from the archive (line 3). In the main loop (line 5-24), DMS regresses the discount model to match a dataset $\mathcal{D}_A$ of input measure values $\boldsymbol{s}$ and their corresponding discount value targets t_A. The dataset entries $(\boldsymbol{s}, t_A)$ come from two sources. The first source is solutions sampled by the emitters (line 12-13). For each emitter solution $\boldsymbol{\theta}$, DMS creates an entry with the solution's measure values $\boldsymbol{s} = \boldsymbol{m}(\boldsymbol{\theta})$ and a target t_A that reproduces CMA-MAE's threshold update rule (Sec. 6.3):

$$t_A = \begin{cases} \hat{f}_A(\boldsymbol{s}) & \text{if } f(\boldsymbol{\theta}) \leq \hat{f}_A(\boldsymbol{s}) \\ (1-\alpha)\hat{f}_A(\boldsymbol{s}) + \alpha f(\boldsymbol{\theta}) & \text{if } f(\boldsymbol{\theta}) > \hat{f}_A(\boldsymbol{s}) \end{cases} \tag{6.1}$$

Here, if the objective value $f(\boldsymbol{\theta})$ is worse than its discount value $\hat{f}_A(\boldsymbol{s})$, t_A is set to the current discount value. If the objective value exceeds the discount value, t_A is a linear combination between the objective value and the current discount value, weighted by archive learning rate α. Similar to CMA-MAE's threshold update rule, this target aims to slowly increase the discount values (and hence decrease the improvement values) in areas of the measure space that DMS has explored, so that the emitters are required to find solutions in new areas of measure space.

The second source of data for $\mathcal{D}_A$ is "empty points," i.e., the centers of unoccupied archive cells. During preliminary experiments, we noticed that updating the discount model caused it to change its outputs in areas of the measure space that were not represented in the dataset $\mathcal{D}_A$. In particular, in areas that had not been explored yet, the discount model should have output the minimum objective f_{min}, but it instead output high arbitrary values. To prevent this issue by "clamping down" the discount model in unexplored areas of the archive, we sample n_{empty} unoccupied cells from the archive (line 21). We then add the center of each cell to the dataset $\mathcal{D}_A$, with an associated target of $t_A = f_{min}$ (line 22-23). If there are fewer than n_{empty} unoccupied archive cells, we select all such cells. Note that in the CVT archive, the "center" of the cell is that cell's centroid.

Summary. DMS performs two phases. First, it searches for solutions that improve the archive $\mathcal{A}$ by sampling solutions with the emitters. Since the emitters contain CMA-ES instances, solutions are sampled from a Gaussian (line 9). As DMS progresses, each emitter updates its Gaussian based on archive improvement rankings (line 18) computed via the discount model (line 11). If CMA-ES converges, the emitters reset (line 19-20). Second, DMS trains the discount model $\hat{f_A}$, ensuring improvement values remain accurate. It collects the dataset $\mathcal{D}_A$ (line 12-13, 21-23) and regresses $\hat{f_A}$ to match these values (line 24). Thus, $\hat{f_A}$ guides the emitters to fill a high-performing archive.

6.6 Domains

We evaluate DMS on standard QD benchmarks and in a setting we refer to as Quality Diversity with Datasets of Measures (QDDM).

6.6.1 Benchmarks

Linear Projection (LP) [77] benchmarks distortion by creating a measure function that projects the majority of an n-dimensional solution space into the center of a k-dimensional measure space. We set $n = 100$ and instantiate LP with the Sphere, Rastrigin, and Flat objectives. With the Flat objective [142], which only outputs 1.0, LP becomes a *diversity optimization* domain where solutions differ solely by their measures. We use six instantiations of LP, named by the measure space dimensionality k and the objective function: **2D LP (Sphere)**, **10D LP (Sphere)**, **2D LP (Rastrigin)**, **10D LP (Rastrigin)**, **2D LP (Flat)**, **10D LP (Flat)**.

Arm Repertoire [249] is an inverse kinematics domain where solutions are joint angle configurations of a 2D planar arm with $n = 100$ joints. The measure function outputs the 2D position of the arm's end effector. The objective indicates the variance of the n joint angles.

6.6.2 Quality Diversity with Datasets of Measures

We propose the QDDM setting, where instead of designing measure functions, a user provides a dataset indicating their desired measure values. The defining feature of QDDM is that the user provides high-dimensional data, e.g., images, audio, or text, and the measure space S is the space of such data, e.g., $S \subseteq \mathbb{R}^{k=256*256*3=196,608}$ if the data are $256 \times 256 \times 3$ RGB images.

Initially, it seems problematic to construct an archive for QDDM due to the high dimensionality of the measure space. However, we adopt the manifold hypothesis [66], i.e., the assumption that most high-dimensional data lie on a low-dimensional manifold embedded within the high-dimensional space. We recognize that the distribution of measures relevant to a user occupies only a small region of the overall measure space, and this distribution is reflected in the user's dataset. Hence, we propose to construct a CVT archive where the centroids are the points in the dataset. Here, the CVT no longer uniformly partitions the measure space. Rather, *it only partitions the small region of measure space desired by the user and indicated in the dataset.* However, the CVT archive introduces a new consideration for QDDM, viz., the

choice of *distance function*. To locate the cell where a solution belongs, CVT archives find the centroid closest to the solution's measures. While Euclidean distance is common here, it is not always ideal [248], which may be especially true when the measures are as high-dimensional as images or text.

Triangle Arrangement (TA). We introduce two QDDM domains. First, TA builds on computational creativity domains [237] and involves arranging a prespecified number of triangles to create images. A solution consists of the vertices, brightness (we consider grayscale images), and alpha (transparency) of each triangle. A solution's measure is created by rendering the triangles as a raster image. We specify desired images (measures) by sampling 1000 images from either MNIST [141] or Fashion MNIST [263], leading to two versions of this domain: **TA (MNIST)** and **TA (F-MNIST)**. Since the images in these datasets are 28×28, the measure space is 784-dimensional. Drawing from the loss function in prior work [237], we define the distance function as Euclidean distance, so each solution is placed in the archive cell of the digit it most resembles. To make the triangle images resemble the MNIST images, we define the objective as the negative (to facilitate maximization) mean squared error between the triangle image and the archive centroid (MNIST image) to which it is assigned.

Latent Space Illumination (LSI). LSI entails exploring the latent space of a generative model to create images with diverse measures. While prior work [75] considers LSI with low-dimensional measures, we consider LSI where the measures are images. As described in Sec. 6.1, we search for images of hikers in different landscapes. The solutions are latent vectors (w-space, but not w^+-space) of StyleGAN3 [127], with images of faces output by StyleGAN3 serving as measures. Thus, the measure space is the space of $256 \times 256 \times 3$ images. The desired measures are specified with a dataset of 10,000 landscape images sampled from LHQ256 [225]. To associate hikers with landscapes in the CVT archive, the distance function is the CLIP score [193] between the face and landscape; CLIP score is more semantically meaningful than Euclidean distance. The objective is the CLIP score between the face image and the prompt "A photo of the face of a hiker." We also add a regularizer loss [72] to penalize latent vectors that fall outside the StyleGAN3 training distribution, and similar to TA, we reward higher alignment between faces and landscapes by adding the CLIP score between the face and the landscape to which it is assigned. We refer to this domain as **LSI (Hiker)**.

Table 6.1: Mean QD Score ("QD") and Coverage ("Cov") for each algorithm in each domain.

	2D LP (Sphere)		10D LP (Sphere)		2D LP (Rastrigin)		10D LP (Rastrigin)		2D LP (Flat)	
	QD	Cov	QD	Cov	QD	Cov	QD	Cov	QD	Cov
DMS	**6,978.20**	**95.89%**	**6,409.50**	**89.21%**	**5,738.90**	**91.67%**	**5,138.81**	**88.19%**	**7,902.05**	**79.02%**
CMA-MAE	6,327.90	80.95%	608.53	6.95%	5,258.59	80.14%	246.55	2.98%	5,675.90	56.76%
DDS	3,156.24	70.75%	4,237.72	60.07%	2,495.11	71.68%	3,331.70	59.54%	6,967.75	69.68%
MAP-Elites (line)	4,908.81	60.42%	2,570.74	29.20%	3,841.05	56.63%	2,001.76	28.04%	4,510.65	45.11%
MAP-Elites	4,163.41	50.76%	228.65	2.35%	3,172.59	48.21%	499.66	7.09%	4,327.00	43.27%

	10D LP (Flat)		Arm Repertoire		TA (MNIST)		TA (F-MNIST)		LSI (Hiker)	
	QD	Cov	QD	Cov	QD	Cov	QD	Cov	QD	Cov
DMS	**7,982.15**	**79.82%**	**7,963.44**	80.15%	951.56	**99.84%**	**701.14**	**72.28%**	**214.91**	3.77%
CMA-MAE	1,554.90	15.55%	7,902.43	79.22%	**954.27**	99.48%	625.65	63.92%	14.61	1.56%
DDS	6,004.95	60.05%	5,568.23	**80.24%**	—	—	—	—	—	—
MAP-Elites (line)	757.75	7.58%	7,458.67	75.60%	945.60	98.86%	551.13	56.68%	-51,827.44	**7.49%**
MAP-Elites	125.65	1.26%	7,411.10	75.42%	941.94	98.42%	513.13	52.68%	-18,917.87	5.06%

6.7 Experiments

We evaluate DMS in low- and high-dimensional measure spaces through experiments in 7 benchmarks [2D LP (Sphere), 10D LP (Sphere), 2D LP (Rastrigin), 10D LP (Rastrigin), 2D LP (Flat), 10D LP (Flat), Arm Repertoire] and 3 QDDM domains [TA (MNIST), TA (F-MNIST), LSI (Hiker)]. In each domain, we conduct a between-groups study with the algorithm as the independent variable: besides DMS, we consider the black-box QD algorithms described in Sec. 6.3: CMA-MAE, DDS, MAP-Elites (line), and MAP-Elites. We consider two dependent variables. *QD Score* [190] represents overall performance by summing the objectives of all solutions in the archive, as is done in the QD objective (Sec. 6.2). *Coverage* indicates how much of the measure space has been explored by computing the percentage of archive cells that have a solution in them. Note that the objective is normalized to have a maximum value of 1 in all domains. *Our hypothesis is that DMS will outperform all other algorithms in both QD Score and Coverage.* We implement all algorithms with pyribs [238]; hyperparameters are in Sec. 6.8.

6.7.1 Analysis

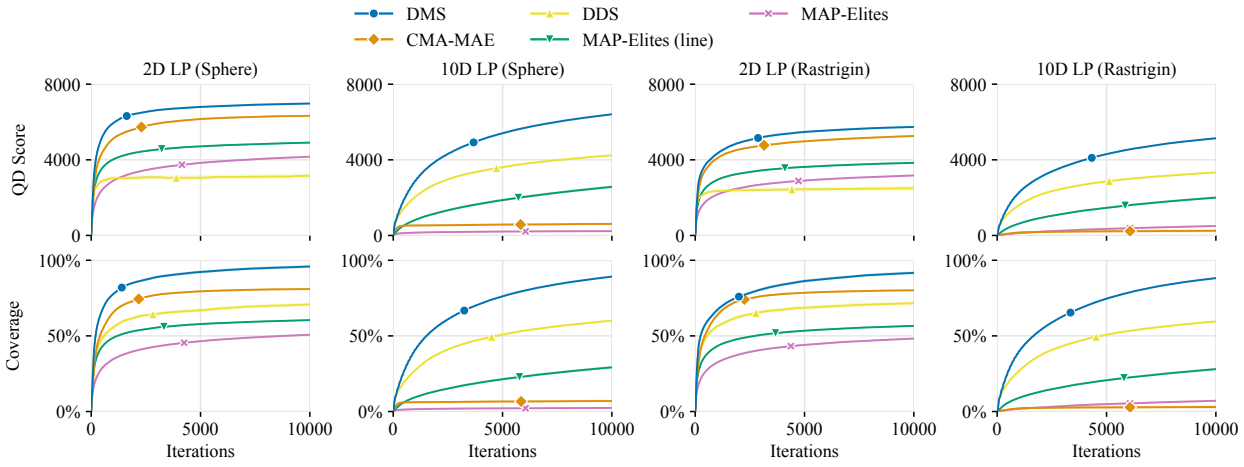
Table 6.1 summarizes the results from 20 trials in the benchmark domains and 5 trials in the QDDM domains. Fig. 6.2 shows sample images from DMS in LSI (Hiker). We could not run DDS in the QDDM domains due to the KDE's runtime, which grows linearly with measure space dimensionality. Visual inspection showed the results were normally distributed, but Levene's test showed most settings were non-homoscedastic. Thus, to analyze the results, we conducted Welch's one-way ANOVA in each domain for each dependent variable. All ANOVAs were significant ($p < 0.001$), so we performed pairwise comparisons with the Games-Howell test. We additionally include the following results:

- Fig. 6.3 and Fig. 6.4 show the mean and standard error of the mean of both dependent variables (QD Score and Coverage), for all algorithms in all domains. The plots shows the values over 10,000 iterations, and the table shows the final values. Note that since the objective is always 1.0 in the LP (Flat) domains, the QD Score and Coverage differ by a factor of the number of cells in the archive, i.e., the QD Score is 10,000 times the Coverage. In the plot for LSI (Hiker), the QD Score is initially negative since the algorithms receive a regularization penalty due to generating images outside the training distribution of StyleGAN3. Neither MAP-Elites variant's QD Score is visible due to being large negative values.

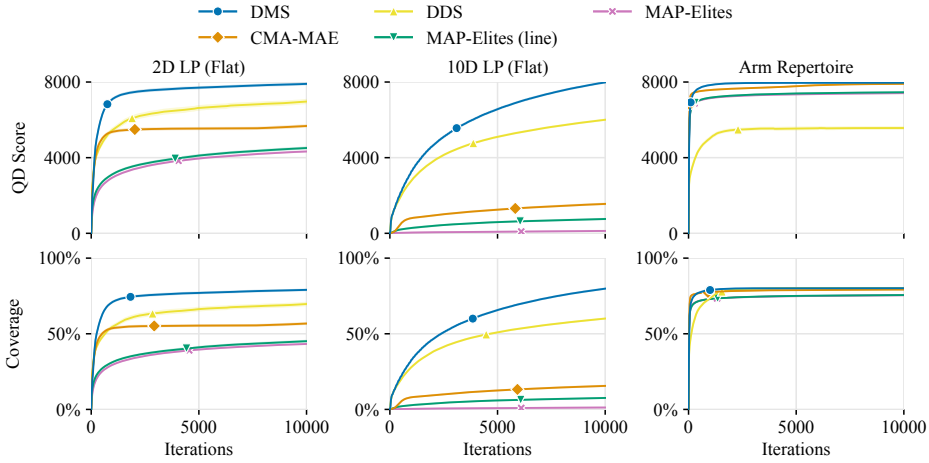
- Fig. 6.5 and Fig. 6.6 show sample images from DMS in the TA (MNIST) and TA (F-MNIST) domains. Note that by default, these domains render the triangles into a 28×28 image. However, since the triangles in each solution form a vector graphic, they can be rendered at any resolution. Thus, for visualization purposes, we rendered them at 280×280 resolution in these figures.

- Fig. 6.7 shows heatmaps of a randomly selected archive of each algorithm, in domains with 2D measure spaces. Fig. 6.8 shows how the archive and discount model in DMS progresses over iterations in the 2D LP (Sphere) benchmark. We include further descriptions and analyses of both of these figures in their captions.

Figure 6.3: Mean and standard error of the mean for QD Score and Coverage of each algorithm in each domain. Standard error may not be visible in some plots.

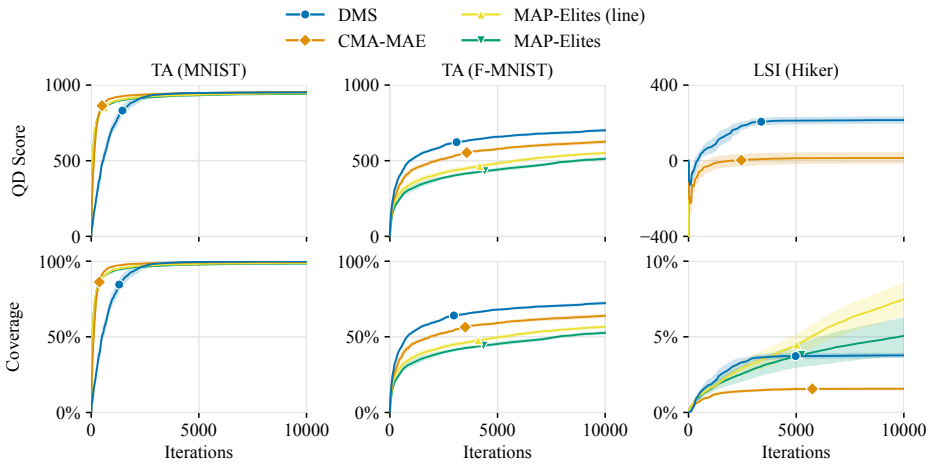


	2D LP (Sphere)		10D LP (Sphere)		2D LP (Rastrigin)		10D LP (Rastrigin)	
	QD Score	Coverage	QD Score	Coverage	QD Score	Coverage	QD Score	Coverage
DMS	**6,978.20 ±17.94**	**95.89 ±0.43%**	**6,409.50 ±13.50**	**89.21 ±0.18%**	**5,738.90 ±19.78**	**91.67 ±0.44%**	**5,138.81 ±10.31**	**88.19 ±0.12%**
CMA-MAE	6,327.90 ±3.60	80.95 ±0.06%	608.53 ±24.78	6.95 ±0.31%	5,258.59 ±5.97	80.14 ±0.11%	246.55 ±36.93	2.98 ±0.49%
DDS	3,156.24 ±91.07	70.75 ±0.65%	4,237.72 ±21.18	60.07 ±0.28%	2,495.11 ±93.14	71.68 ±0.96%	3,331.70 ±18.20	59.54 ±0.27%
MAP-Elites (line)	4,908.81 ±2.48	60.42 ±0.04%	2,570.74 ±39.16	29.20 ±0.45%	3,841.05 ±4.67	56.63 ±0.07%	2,001.76 ±23.09	28.04 ±0.33%
MAP-Elites	4,163.41 ±4.10	50.76 ±0.07%	228.65 ±8.79	2.35 ±0.09%	3,172.59 ±8.43	48.21 ±0.14%	499.66 ±14.06	7.09 ±0.22%



	2D LP (Flat)		10D LP (Flat)		Arm Repertoire	
	QD Score	Coverage	QD Score	Coverage	QD Score	Coverage
DMS	**7,902.05 ±35.68**	**79.02 ±0.36%**	**7,982.15 ±24.43**	**79.82 ±0.24%**	**7,963.44 ±2.47**	80.15 ±0.01%
CMA-MAE	5,675.90 ±46.04	56.76 ±0.46%	1,554.90 ±26.04	15.55 ±0.26%	7,902.43 ±1.93	79.22 ±0.02%
DDS	6,967.75 ±98.80	69.68 ±0.99%	6,004.95 ±21.52	60.05 ±0.22%	5,568.23 ±54.86	**80.24 ±0.00%**
MAP-Elites (line)	4,510.65 ±41.43	45.11 ±0.41%	757.75 ±20.10	7.58 ±0.20%	7,458.67 ±2.56	75.60 ±0.03%
MAP-Elites	4,327.00 ±17.29	43.27 ±0.17%	125.65 ±15.88	1.26 ±0.16%	7,411.10 ±2.56	75.42 ±0.03%

Figure 6.4: Mean and standard error of the mean for QD Score and Coverage of each algorithm in each domain. Standard error may not be visible in some plots.



	TA (MNIST)		TA (F-MNIST)		LSI (Hiker)	
	QD Score	Coverage	QD Score	Coverage	QD Score	Coverage
DMS	951.56 ±0.95	**99.84 ±0.10%**	**701.14 ±5.65**	**72.28 ±0.60%**	**214.91 ±15.23**	3.77 ±0.13%
CMA-MAE	**954.27 ±0.69**	99.48 ±0.09%	625.65 ±8.60	63.92 ±0.88%	14.61 ±26.18	1.56 ±0.05%
DDS	—	—	—	—	—	—
MAP-Elites (line)	945.60 ±1.38	98.86 ±0.15%	551.13 ±6.16	56.68 ±0.64%	-51,827.44 ±17953.11	**7.49 ±1.03%**
MAP-Elites	941.94 ±2.01	98.42 ±0.22%	513.13 ±6.80	52.68 ±0.70%	-18,917.87 ±5705.28	5.06 ±1.17%

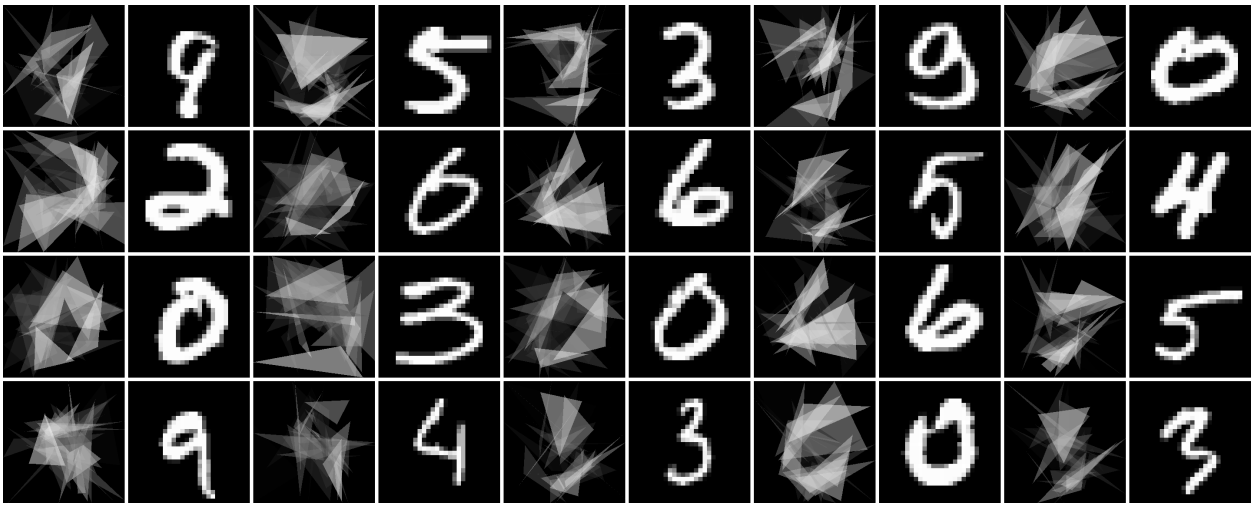


Figure 6.5: A random subset of images generated by DMS in the TA (MNIST) domain, where desired measures are sampled from the MNIST dataset. The goal in this domain is to arrange triangles to look like the given MNIST digits. Each rendered triangle image is shown to the left of its corresponding MNIST digit.

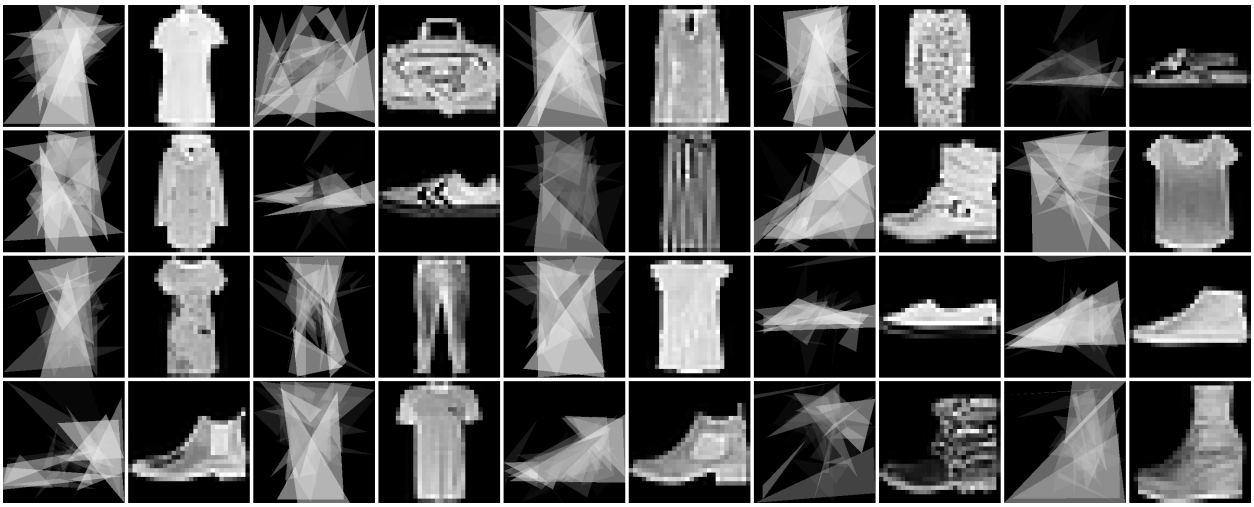


Figure 6.6: A random subset of images generated by DMS in the TA (F-MNIST) domain, where desired measures are sampled from the Fashion MNIST dataset. The goal in this domain is to arrange triangles to look like the images of fashion items. Each rendered triangle image is shown to the left of its corresponding fashion item.

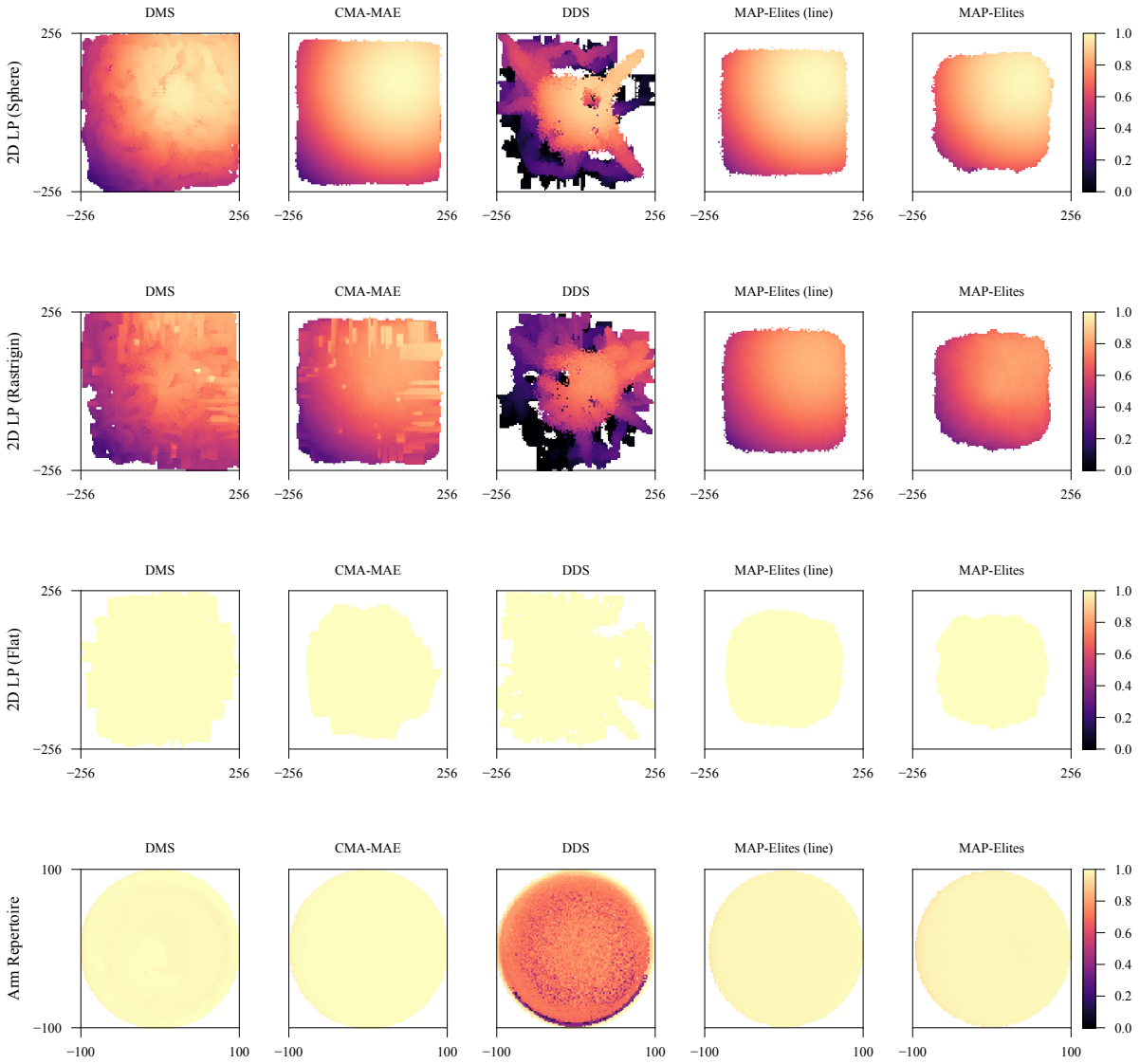


Figure 6.7: Heatmaps of a randomly selected archive produced by each algorithm in domains with 2D measure spaces. Each row contains heatmaps for a single domain. The axes of the heatmaps are the measures, while the color of each cell indicates the objective value. Notably, the heatmaps show how DMS achieves high coverage of the measure space. They also show how DDS achieves good coverage but cannot achieve high objective values since it is a diversity optimization algorithm.

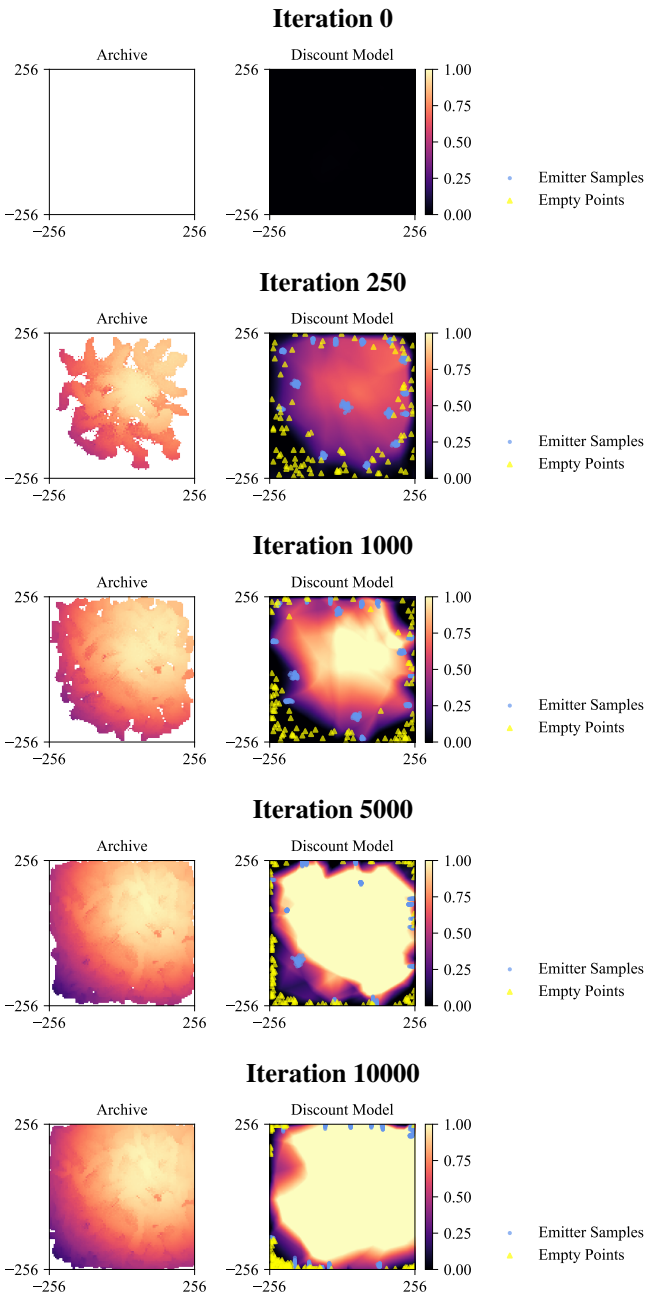


Figure 6.8: Progression of the archive and discount model in DMS in the 2D LP (Sphere) benchmark. The left heatmap shows the archive, while the right heatmap shows the discount model. To plot the discount model, we computed its output at points in a 200×200 grid in measure space. The discount model heatmap also shows the dataset $\mathcal{D}_A$ of points on a given iteration — blue circles indicates points created with solutions from the emitters, and yellow triangles indicate empty points. On Iteration 0, the discount model initializes to output f_{min} everywhere. On Iteration 250, as the emitters begin to populate the archive, the discount model begins to output higher values in areas that have been explored. However, unexplored areas still maintain low values (shown as dark colors) due to the empty points in the dataset. On further iterations, the discount model outputs higher and higher values as the emitters populate the archive further, until it outputs high values nearly everywhere on Iteration 10000.

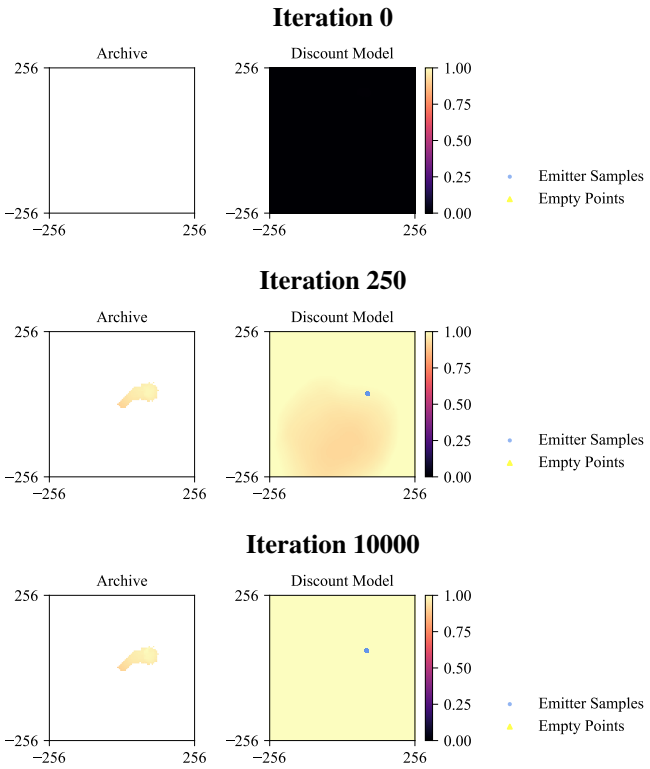


Figure 6.9: Similar to Fig. 6.8, this figure shows how the archive and discount model in DMS progress across iterations. However, this time, DMS does not train the discount model with any empty points, i.e., $n_{empty} = 0$. As a result, the discount model takes on arbitrary values in areas of the measure space that have not been explored yet, as evinced by the high values across the discount model heatmap on Iteration 250 and 10000. Because the discount values are high everywhere, the emitters in DMS mistakenly believe they have explored all areas of the measure space, even though the archive is essentially empty.

Benchmark Domains. In the benchmarks, DMS significantly outperformed all baselines in QD Score and Coverage, except in Arm Repertoire, where DDS had significantly better coverage. The high performance on LP shows that DMS better overcomes distortion than previous algorithms, as these domains are designed to benchmark distortion. Since DDS is a diversity optimization algorithm, we expect it to achieve the best coverage in all domains, so it is surprising that DMS outperforms it in almost all domains, even 2D and 10D LP (Flat), which are diversity optimization domains where the objective is always 1.0.

TA Domains. In TA (MNIST), for both metrics, DMS significantly outperformed the two MAP-Elites algorithms, but there was no significant difference with CMA-MAE. In TA (F-MNIST), DMS significantly outperformed all baselines in both metrics. The coverage results in TA (MNIST) illustrate that not all QDDM domains exhibit high distortion — low distortion is reflected in how all algorithms achieve nearly perfect coverage in TA (MNIST). We believe the high coverage stems from how MNIST images are fairly similar in appearance. As such, an algorithm can fill the archive by generating triangle images that differ only slightly from one another. TA (F-MNIST) seems more challenging, as no algorithm achieves perfect coverage there.

On the other hand, the difficulty of TA (MNIST) seems to lie in optimization of the objective, as the difference in QD Score between DMS/CMA-MAE and the two MAP-Elites algorithms is small yet statistically significant. This property suggests a potential limitation of DMS. By nature of being a model, the discount model in DMS exhibits errors, which we can imagine as adding noise to discount values. We speculate that in domains where objective optimization is less important, the noise is small enough that improvement rankings remain unaffected. However, in a domain that requires fine optimization of the objective like TA (MNIST), this noise interferes with improvement rankings, hindering DMS. CMA-MAE maintains exact values in its histogram and would not have such noise, potentially explaining why DMS does not outperform CMA-MAE's QD Score here.

LSI (Hiker). The results in LSI (Hiker) highlight the difficulty of complex QDDM domains. Here, DMS significantly outperformed CMA-MAE in both metrics, but there was no significant difference with the two MAP-Elites algorithms. While DMS outperforms CMA-MAE, it covers only 3.77% of the archive, although this still represents 377 hiker images. We note that the two MAP-Elites algorithms receive large negative QD Scores and high coverages by generating latent vectors far outside the training distribution of StyleGAN3 and incurring large regularization losses. Similarly, they exhibit high performance variance, so they have no significant difference with DMS.

Ablation Study. We conducted an ablation study of the hyperparameters of DMS. We find that the archive learning rate α behaves similarly as in CMA-MAE: intermediate values enable DMS to balance optimization of the objective and exploration of the measure space, while $\alpha = 0$ causes DMS to over-emphasize the objective. Meanwhile, the "empty points" are necessary for training the discount model: removing them by setting $n_{empty} = 0$ causes performance to drop since the discount model takes on arbitrary values in areas of the measure space that have not been explored (Fig. 6.9). In contrast, setting $n_{empty} = 10, 100,$ or 1000 resolves this issue by "clamping down" the discount model (Fig. 6.8). Overall, our experimental results show how the discount model successfully guides optimization in DMS, leading to high performance across domains with different measure spaces.

6.8 Hyperparameters

Table 6.2 lists the hyperparameters of DMS and all baseline algorithms. All algorithms run for 10,000 iterations. The number of solutions generated and evaluated on each iteration is equal across all algorithms. In DMS, CMA-MAE, and DDS, this number is equivalent to the number of emitters W times the emitter batch size λ. In the two MAP-Elites algorithms, this number is equivalent to the batch size λ. For DDS, we use the KDE version ("DDS-KDE") [142]. In all domains, the objective is normalized to be between 0 and 1, so the minimum objective f_{min} is set to 0. For the benchmark domains, parameters for the baselines are adapted from prior work [72, 142].

Archive. In each domain, all algorithms use the same archive configuration. In benchmark domains, the archive has 10,000 cells, arranged as a 100×100 grid for domains with 2D measure spaces: 2D LP (Sphere), 2D LP (Rastrigin), 2D LP (Flat), Arm Repertoire. The cells are arranged as a 10,000-cell CVT archive for domains with 10D measure spaces: 10D LP (Sphere), 10D LP (Rastrigin), 10D LP (Flat). In QDDM domains, the archive is a CVT archive consisting of centroids sampled from the dataset. There are 1,000 cells in the archive for TA (MNIST) and TA (F-MNIST), and 10,000 cells in the archive for LSI (Hiker). The same CVT is used across all trials of all algorithms per domain (as opposed to randomly regenerating the CVT every trial).

Restart Rule. The restart rule refers to the conditions upon which emitters are restarted from solutions in the archive. "No Imp." refers to restarting when the emitter no longer discovers solutions that are added to the archive (i.e., solutions that improve the archive) [77]. "Basic" refers to restarting only when default CMA-ES [108] termination rules are met. An integer value R refers to restarting every R iterations [72].

Table 6.2: Hyperparameters.

	2D LP (Sphere)	10D LP (Sphere)	2D LP (Rastrigin)	10D LP (Rastrigin)	2D LP (Flat)	10D LP (Flat)	Arm Repertoire	TA (MNIST)	TA (F-MNIST)	LSI (Hiker)
DMS										
Number of emitters W	15	15	15	15	15	15	15	5	5	1
Emitter batch size λ	36	36	36	36	36	36	36	36	36	36
Initial step size σ_0	0.5	0.5	0.5	0.5	0.5	0.5	0.2	0.1	0.1	0.02
Archive learning rate α	0.1	0.1	0.1	0.1	0.1	0.1	0.001	0.001	0.1	1.0
Restart rule	Basic	100	Basic	100	Basic	100	Basic	50	50	Basic
Selection rule	μ	μ	μ	μ	μ	μ	μ	μ	μ	μ
Empty points n_{empty}	100	100	100	100	100	100	100	100	100	100
Initial points n_{init}	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
CMA-MAE										
Number of emitters W	15	15	15	15	15	15	15	5	5	1
Emitter batch size λ	36	36	36	36	36	36	36	36	36	36
Initial step size σ_0	0.5	0.5	0.5	0.5	0.5	0.5	0.2	0.1	0.1	0.02
Archive learning rate α	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.001	0.1	0.01
Restart rule	Basic	Basic	Basic	Basic	Basic	Basic	Basic	50	50	Basic
Selection rule	μ	μ	μ	μ	μ	μ	μ	μ	μ	μ
DDS										
Number of emitters W	15	15	15	15	15	15	15	—	—	—
Emitter batch size λ	36	36	36	36	36	36	36	—	—	—
Initial step size σ_0	1.5	1.5	1.5	1.5	1.5	1.5	0.5	—	—	—
Bandwidth h	25.6	5.12	25.6	5.12	25.6	5.12	10.0	—	—	—
Buffer size	10,000	10,000	10,000	10,000	10,000	10,000	10,000	—	—	—
Restart rule	No Imp.	No Imp.	No Imp.	No Imp.	No Imp.	No Imp.	No Imp.	—	—	—
Selection rule	Filter	Filter	Filter	Filter	Filter	Filter	Filter	—	—	—
MAP-Elites (line)										
λ (batch size)	540	540	540	540	540	540	540	180	180	36
σ_1	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.1	0.1	0.1
σ_2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
MAP-Elites										
λ (batch size)	540	540	540	540	540	540	540	180	180	36
σ	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.1	0.1	0.1

Discount Model Architecture and Training. In all domains except for LSI (Hiker), the discount model is a three-layer MLP with layer sizes $[k, 128, 128, 1]$, where k is the dimensionality of the measure space. In LSI (Hiker), the architecture differs slightly in that measures are embedded with CLIP [193] before being passed into a three-layer MLP with layer sizes $[512, 128, 128, 1]$. Beyond that, the details of all MLPs are identical. There is ReLU activation after every layer except the output layer. Inputs to the network are normalized to the range $[-1, 1]$ based on the bounds of the feature space; in the case of images, these bounds are assumed to be $[0, 1]$. Networks are instantiated with the

default PyTorch initialization. The discount models are trained with an Adam optimizer with settings of learning rate $\alpha = 0.001$ and $\beta_1 = 0.9$, $\beta_2 = 0.999$. The loss function is mean squared error (MSE), and we train with a batch size of 32. Each iteration (including during the initial training of the model to output f_{min}), the model trains until an average cutoff loss of at most 0.05 is reached over the whole dataset $\mathcal{D}_A$, with a maximum of five epochs allowed. In practice, we found that training almost always required only one epoch to reach a cutoff loss of 0.05. The optimizer is maintained throughout the entire run.

6.9 Related Work

Quality Diversity Optimization. Applications of QD include robotics [49, 118, 165], drug discovery [21, 251], urban planning [86], and finance [88]. In computer vision, QD can create diverse images by exploring the latent space of generative models, e.g., StyleGAN [128] in our work and in [73], and Stable Diffusion [198] in [58]. Such images can form a synthetic dataset for debiasing downstream models [34]. In reinforcement learning, QD can generate diverse locomotion policies [175, 186, 14, 33, 264], and in red-teaming, QD can probe a large language model (LLM) to produce harmful outputs [206].

Our work fits a growing trend of integrating models into QD. To accelerate evaluations, multiple works [19, 18, 130, 267, 83, 105] train surrogate models to approximate expensive objectives and/or measures. Others [96, 97, 204] build models that guide the creation of new solutions, especially in reinforcement learning [239, 175, 186, 14, 33, 100]. Furthermore, whereas DMS searches directly in the space of high-dimensional measures, several approaches [46, 180, 99, 158] build models that compress high-dimensional measures into low-dimensional measures. Finally, as discussed in Sec. 6.6.2, we apply the manifold hypothesis [66] in *measure space* when creating the archive in QDDM domains. Prior work [249, 84, 195, 114] applies the manifold hypothesis in *solution space* by searching over the *elite hypervolume*, a low-dimensional manifold where the solutions to each QD problem are hypothesized to exist.

Related to our proposed QDDM setting, developing intuitive ways to specify measures of diversity is an active research area in QD. For example, QD through AI Feedback (QDAIF) [26] evaluates measures by querying LLMs for feedback, while the LSI domain in prior work [73] computes the CLIP score between text prompts and generated images. QD through Human Feedback (QDHF) [58] learns measures from human preferences via a contrastive loss. Each method's suitability depends on which user effort is easiest. For example, QDAIF excels when measures can be

conveniently specified to an LLM evaluator that outputs a vector of measures. Conversely, QDAIF can be limited by the challenges of prompt engineering and the stochasticity of LLM outputs. Moreover, it may be difficult to create vector-valued measures that elicit the desired diversity, like in LSI (Hiker), where QDDM makes it easy to specify "where a hiker is located" with landscape images. A key distinction is that QDDM specifies desired measure *values*, while the above methods all define measure *functions*. If an appropriate dataset does not exist, designing a measure function may be more appropriate, since datasets require significant effort to curate. On the other hand, we believe QDDM will be applicable in many problems since datasets are abundant in machine learning.

Computational Creativity. Our TA domains draw from prior work [237] that arranges triangles to represent images and text prompts with evolution strategies. Similarly, various works arrange basic shapes to create artistic images [177, 16, 30, 70]. To our knowledge, QD algorithms have not been applied in this setting, but they have generated other forms of art, such as line drawings [158] and images [265].

Latent Space Exploration. LSI (Hiker) is an example of *latent space illumination*, where a QD algorithm searches for latent vectors that elicit diverse, high-performing outputs of a generative model. LSI was first introduced to generate video game levels [75, 74, 207, 213, 230] and 2D shapes [104]. Later work [73, 72] explored the latent space of StyleGAN [128] to generate celebrity images with low-dimensional (2D) measures based on the CLIP score [193, 116]. Beyond LSI, various methods aim to navigate latent spaces. Several works discover interpretable directions in GAN latent spaces, where these directions manipulate pose or facial features [221, 219, 254, 220] or transform position and scale [187]. In single-objective optimization, prior works search the latent spaces of generative models to create video game levels [253, 159, 235] or synthetic fingerprints [22] that satisfy desired characteristics.

6.10 Conclusion

By searching in distorted and high-dimensional measure spaces, DMS offers two benefits for QD practitioners. First, DMS can improve performance in current QD applications. As the results in the benchmark domains show (Sec. 6.7.1), DMS outperforms current algorithms in various domains with low-dimensional measure spaces. Since most current applications involve low-dimensional measure spaces, we believe DMS will also outperform current QD algorithms in current applications. Second, DMS enables *new* applications by addressing the proposed QDDM setup (Sec. 6.6.2), where diversity in a high-dimensional measure space like images is specified by providing a dataset. We believe framing

measures in terms of datasets makes QD more accessible by not only alleviating the need to hand-design measure functions, but also making it possible to specify measures that cannot easily be represented by low-dimensional values. For example, as the TA and LSI (Hiker) domains showed, we can now specify the measure space in vision and art domains with image datasets. Overall, given that datasets are central to machine learning, we believe it will prove fruitful to frame problems across machine learning as QDDM problems and solve them with algorithms like DMS.

DMS also has the potential for negative societal impacts. For example, its abilities in QDDM settings may exacerbate biases in the dataset of measures, image generator [127], and even distance metric [193]. These biases can be mitigated by carefully managing all datasets, including that used to train the generator [185, 156]. In general, QD can also debias models by generating balanced datasets [34]. In addition, using QDDM methods to drive generative search towards specific artistic styles defined in the measure space can discount unique styles created by human artists [218]. Alternative objective functions [246] may enable finding solutions that reflect the dataset of measures without mimicking artists' creativity.

Our work has several limitations. First, while searching over the discount model garners high performance, training it induces computational overhead. Second, while DMS trains the discount model with targets that reproduce CMA-MAE's threshold update, alternative targets may improve properties like smoothness. Finally, we primarily consider small MLPs for the discount model. We are excited for future work in domains that require more advanced discount model architectures.

Chapter 7

pyribs: A Bare-Bones Python Library for Quality Diversity Optimization

7.1 Introduction

Many research problems decompose into highly contextual components that prevent one solution from working well across all possible situations. In such cases, developing a set of solutions rather than a single solution enables researchers to account for a range of contexts. For instance, a roboticist may develop diverse walking gaits so that their robot can adapt to different morphological considerations [49], while a video game designer may generate multiple video game levels so that players can experience various levels of difficulty [75, 61], and a chemist may create multiple viable drug candidates which exhibit unique properties [251].

Quality diversity (QD) optimization [35] addresses such problems by searching for collections of diverse, high-performing solutions. Originating in neuroevolution with Novelty Search [145, 146] and MAP-Elites [169], QD has grown to become a general-purpose optimization paradigm with applications in a number of areas. As of writing, there are at least 167 papers on the topic [48], spanning areas as diverse as reinforcement learning [175, 42, 239, 186, 240, 44], robot manipulation [165, 164], human-robot interaction [71, 76, 74], video game level generation [75, 61], agent testing [19], generative modeling [73], urban planning [86], design [83], internet congestion control [64], and drug discovery [251]. QD has also moved outside of publications and into more popular forms of media like blog posts [268, 119, 79, 162, 69, 5, 260] and conference tutorials [51, 52, 53, 40].

To grow further, we believe the QD community must overcome two challenges. The first challenge is to develop a conceptual framework capable of implementing the wide and growing range of QD algorithms. Many QD algorithms contain interchangeable components, and a unified framework allows for mixing several state-of-the-art components

into new algorithms as the field advances. To this end, previous work has proposed the Unifying Modular Framework (UMF) [50] to connect the two main families of QD algorithms, Novelty Search and MAP-Elites. However, UMF was primarily designed for QD algorithms based on genetic operators [54], which limits its applicability to recently developed QD algorithms that have a strong optimization component, such as algorithms which incorporate Evolution Strategies (ES) [77, 44, 240, 239, 42], gradient ascent [73, 72], or Bayesian Optimization [130].

The second challenge is to implement this framework in software which can support a wide range of users, ranging from beginners entering the field to experienced researchers seeking to develop new algorithms. Historically, the conception of flexible, well-documented software libraries has been quintessential to the blooming of popular research areas. For instance, PyTorch [183] and TensorFlow [1] have catalyzed the development and deployment of countless deep learning algorithms in academia and industry [112], and pycma [109] has popularized the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) as one of the standard tools of evolutionary computation. Such libraries have profound effects on their respective fields because not only do they provide powerful features concealed with an expressive, user-friendly application programming interface (API), but they also make these features accessible through comprehensive documentation and tutorials, enabling new practitioners to incorporate the latest algorithms into their projects. Thus far, the QD community has introduced a number of its own libraries. While these libraries have successfully spurred research, they are targeted towards researchers *within* the QD community, offering them high performance [167, 150], reference implementations [168], or a rich end-to-end experience [31].

To address these challenges, we have developed the pyribs library, which implements a conceptual framework that we call RIBS.* As shown in Fig. 7.1, a QD algorithm in RIBS is comprised of three components: (1) an *archive* to store solutions generated by the QD algorithm, (2) one or more *emitters* to generate new solutions, and (3) a *scheduler* to manage the interaction of the archive and emitters.

RIBS is highly *modular*: As Table 7.1 shows, *many existing QD algorithms can be composed by replacing individual components of the framework*. The table also highlights unexplored gaps that could be filled by combining different components, indicating potentially promising areas for future research. Yet, the modular design does not sacrifice simplicity, a key feature in attracting new practitioners.

*The name "RIBS" stems from the title of Fontaine et al. [77], "Covariance Matrix Adaptation for the **R**apid **I**llumination of **B**ehavior **S**pace," which introduced the concepts of emitters and schedulers. The name "pyribs" is thus a combination of "Python" and "RIBS." The proper spelling of pyribs is all-lowercase, similar to pycma [109], except at the beginning of sentences, when it is capitalized as Pyribs.

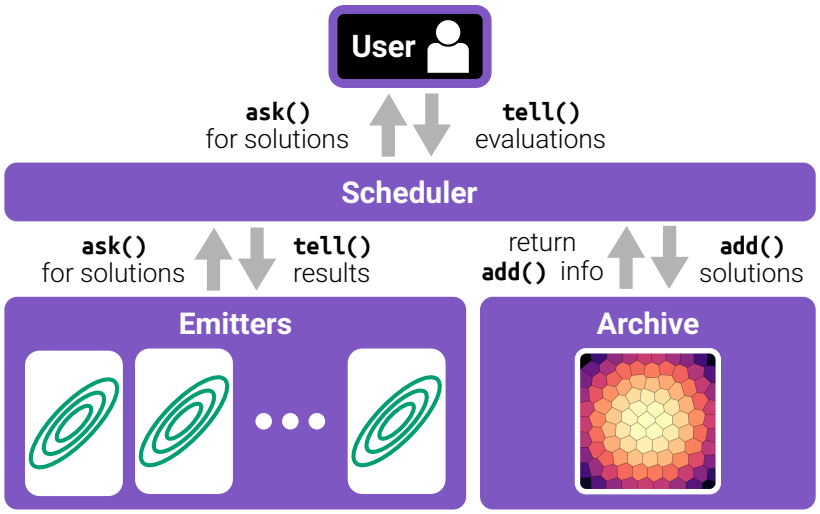


Figure 7.1: Pyribs implements the RIBS framework for QD optimization. The user first `ask()`'s for solutions from a *scheduler*. The scheduler selects *emitters* to `ask()` for solutions and returns the solutions to the user. After evaluating the solutions, the user `tell()`'s the results to the scheduler. The scheduler `add()`'s the solutions to the *archive* and receives information that it `tell()`'s to the emitters, enabling the emitters to update their internal search state.

Moreover, the software implementation of RIBS in pyribs enables seamlessly translating these compositions into code for experimentation and engineering. We achieve this functionality by constructing the library around the following design principles:

Simple: Centered *only* on components that are absolutely necessary to run a QD algorithm, allowing users to combine the framework with other software frameworks.

Flexible: Capable of representing a wide range of current and future QD algorithms, allowing users to easily create or modify components.

Accessible: Easy to install and learn, particularly for beginners with limited computational resources.

Pyribs offers modular components that can be assembled into a QD algorithm and controlled with an API inspired by pycma's ask-tell interface [109]. It also features extensive documentation, including tutorials (Fig. 7.3) demonstrating its usage.[†] Since its inception in 2021, pyribs has grown to support the research of at least a dozen groups across academia and industry worldwide. As of writing, it has been applied to image generation [73, 72], video game level generation [61], environment generation [19], reinforcement learning [240, 239], hyperparameter optimization [211], architecture design [85], and internet congestion control [64].

[†]Website: `https://pyribs.org`, Source Code: `https://github.com/icaros-usc/pyribs`, Documentation and Tutorials: `https://docs.pyribs.org`

Table 7.1: By selecting different components in the RIBS framework, we can compose a variety of recent algorithms from the QD literature and test them in pyribs. Furthermore, we can identify combinations of components which may lead to new algorithms. Refer to Sec. 7.3.2 for more details on the archives, emitters, and schedulers shown here.

	Archive				Emitters					Scheduler	
	Grid	CVT	Sliding Boundaries	Unstructured	Gaussian	Iso+LineDD	CMA-ES	Genetic Algorithm	Gradient Arborescence	Basic	Bandit
MAP-Elites [169]	×				×					×	
CVT-MAP-Elites [248]		×			×					×	
Iso+LineDD MAP-Elites [249]		×				×				×	
MESB [78]			×		×					×	
NSLC [146]				×				×		×	
CMA-ME [77]	×						×			×	
CMA-MAE [72]	×						×			×	
ME-MAP-Elites [47]	×					×	×				×
CMA-MEGA [73]	×								×	×	
CMA-MAEGA [72]	×								×	×	

7.2 Background

7.2.1 Quality Diversity

7.2.1.1 Focus

The pyribs library focuses on continuous optimization problems over the search space $\mathbb{R}^n$, the same class of problems targeted by the pycma [109] library. By focusing only on continuous optimization, the library becomes less abstract as search vectors become explicitly defined. Yet, continuous optimization contains an expressive class of problems that the QD community cares about.

7.2.1.2 Definition

We define the continuous QD problem. We assume an *objective function* $f : \mathbb{R}^n \to \mathbb{R}$ and k *measure functions*[‡] $m_i : \mathbb{R}^n \to \mathbb{R}$, represented jointly as $\boldsymbol{m} : \mathbb{R}^n \to \mathbb{R}^k$. We let $S = \boldsymbol{m}(\mathbb{R}^n)$ be the measure space formed by the range of $\boldsymbol{m}$.

The *QD objective* is to find, for each $\boldsymbol{s} \in S$, a solution $\boldsymbol{\theta} \in \mathbb{R}^n$ such that $\boldsymbol{m}(\boldsymbol{\theta}) = \boldsymbol{s}$ and $f(\boldsymbol{\theta})$ is maximized:

$$\begin{aligned} &\max \quad f(\boldsymbol{\theta}) \\ &\text{subject to} \quad \boldsymbol{m}(\boldsymbol{\theta}) = \boldsymbol{s} \qquad \forall \boldsymbol{s} \in S \end{aligned} \tag{7.1}$$

However, since S is continuous, this objective would require infinite memory to solve, so we relax the QD objective to finding an *archive* (i.e., a finite set) of *representative* solutions $\Theta \subseteq \mathbb{R}^n$.

A special case of the QD problem is the *differentiable quality diversity (DQD)* [73] problem, where the objective and measure functions are first-order differentiable with gradients $\boldsymbol{\nabla} f$ and $\boldsymbol{\nabla} \boldsymbol{m}$.

7.2.1.3 Algorithms

We consider two alternatives of what constitutes a *representative* solution in the QD problem definition (Eq. 7.1), resulting in two families of algorithms.

[‡]Prior work refers to measure function outputs as "behavior characteristics," "behavior descriptors," or "feature descriptors." We use the "measures" terminology in pyribs.

Algorithms based on MAP-Elites [169] tessellate the measure space S into M cells, and Θ is constrained such that each of its solutions falls into a different cell of the tessellation based on its measure values. The vanilla MAP-Elites [169] mutates randomly sampled solutions in the archive with a genetic operator; generated solutions are added to the archive if their objective value exceeds that of the solution currently occupying their corresponding archive cell. Since its inception, MAP-Elites extensions have included new genetic operators, such as the Iso+LineDD operator inspired by crossover [249], as well as new methods for tessellating the measure space to create the archive. For example, MAP-Elites with Sliding Boundaries (MESB) adapts the size of grid cells online to reflect the distribution of solutions in measure space [78], while CVT-MAP-Elites [248] precomputes a centroidal Voronoi tessellation (CVT) [59] of the measure space that defines the archive cells.

Algorithms based on Novelty Search [145, 146] maintain an unstructured archive where each solution must be *novel* by being a certain distance away from its nearest neighbors in measure space. A genetic algorithm then optimizes a population of solutions to achieve further novelty. While Novelty Search itself is a purely diversity-driven approach, many of its successors are designed for QD; for instance, Novelty Search with Local Competition (NSLC) [146] balances between optimizing for the objective and novelty via multi-objective evolutionary algorithms.

QD algorithms have started to incorporate modern optimization algorithms. For example, Covariance Matrix Adaptation MAP-Elites (CMA-ME) [77] directly optimizes for the QD objective with CMA-ES [108]. In QD optimization, it is efficient to search *multiple* regions of the measure space simultaneously, while balancing the exploration of each region. Therefore, CMA-ME introduced the concepts of *emitters* and *schedulers*. Each emitter maintains a separate CMA-ES instance, while the scheduler balances how emitters explore each measure space region. Emitters and schedulers became core components of the RIBS framework (Sec. 7.3). Subsequent works building on CMA-ME include Covariance Matrix Adaptation MAP-Annealing (CMA-MAE) [72], which adds an *archive learning rate* to the MAP-Elites grid archive. The learning rate regulates how quickly a non-stationary discount function changes, resulting in a *soft archive* that balances the tradeoff between pure optimization and exploration. In addition, CMA-MEGA and CMA-MAEGA (CMA-ME / CMA-MAE via a Gradient Arborescence) [73, 72] address DQD problems with similar principles as CMA-ME and CMA-MAE.

Finally, Multi-Emitter MAP-Elites (ME-MAP-Elites) [47] introduced a new scheduler by modifying the method for selecting emitters. While the scheduler in CMA-ME maintains several CMA-ES emitters and a round-robin emitter

scheduler, ME-MAP-Elites maintains an *emitter pool* consisting of emitters from CMA-ME and emitters that apply the Iso+LineDD operator [249]. Every iteration, the scheduler uses a multi-armed bandit selector from prior work [84] to select emitters which are likely to improve the archive.

7.2.2 The Unifying Modular Framework

The Unifying Modular Framework (UMF) [50], an early conceptual QD framework, proposed to unite the components of the two pioneering algorithms in QD optimization: MAP-Elites and NSLC.

In UMF, QD algorithms consist of a *container* — equivalent to a RIBS archive — and a *selector*. On each iteration of a QD algorithm in UMF, the selector generates solutions that are passed through *random variation* (e.g., mutation or crossover), evaluated, and then inserted into the container. Containers include the MAP-Elites grid and NSLC unstructured archive, and selection mechanisms include choosing solutions uniformly at random from the container, as in vanilla MAP-Elites, or selecting from a population as in NSLC.

UMF unified under one framework the two major families of QD algorithms: MAP-Elites and NSLC. However, UMF was proposed when all QD algorithms were based on genetic algorithms, and the framework is not expressive enough to represent modern QD algorithms based on other optimization methods. Specifically, UMF incorporates a selector, which chooses solutions as inputs to genetic operators. While selectors can retain a population of solutions, they are not suitable for optimization algorithms that require an internal state, e.g., an evolution path in CMA-ES [108] or momentum in Adam [136]. Drawing from the architecture of CMA-ME [77], our proposed RIBS framework incorporates emitters, which were designed to encapsulate any optimization algorithm used to generate solutions.

In addition, UMF is not designed to manage multiple populations simultaneously. However, algorithms like CMA-ME and ME-MAP-Elites require this feature to maintain multiple CMA-ES instances. The RIBS framework overcomes this design limitation by incorporating a scheduler, which manages multiple emitters.

7.2.3 Existing QD Libraries

Here we review libraries developed by the QD community, including their goals, features, and relation to pyribs.

7.2.3.1 Sferes$_{v2}$

Sferes$_{v2}$ [167] is a C++ framework for evolutionary computation that also supports QD algorithms. Sferes$_{v2}$ is primarily designed for high performance, leveraging template-based meta-programming to provide an efficient object-oriented interface and offering multi-core parallel execution through Intel TBB and MPI. While the template-based structure results in significant performance benefits, it limits accessibility for non-expert users. In comparison, pyribs focuses solely on QD algorithms rather than on general evolutionary computation. It is a Python-based library that emphasizes accessibility over performance (Sec. 7.4.1.3).

7.2.3.2 QDpy

QDpy [31] is designed to be a feature-rich Python library for QD optimization. Besides supporting ready-to-go implementations of algorithms such as MAP-Elites and CMA-ME, QDpy provides building blocks that can be assembled into new algorithms. To run a QD algorithm, a QDpy user instantiates a *container* (i.e., an archive) and passes it to an *algorithm* object. The user then defines an evaluation function and passes the function to the QDpy system to optimize. QDpy also provides logging and plotting utilities and tools to run the evaluation function on distributed computation.

QDpy's flexibility is limited by the requirement that users pass in an evaluation function. While passing in this function allows users to leverage QDpy's various utilities, this requirement also makes it difficult for users to integrate their own utilities. In contrast, pyribs provides an ask-tell interface where users handle evaluations on their own (Sec. 7.4.2.4). Essentially, pyribs focuses on components necessary for running QD algorithms, allowing users to integrate tools and frameworks with which they are already familiar.

7.2.3.3 pymap_elites

pymap_elites [168] provides customizable reference implementations of MAP-Elites and its variants CVT-MAP-Elites [248], MAP-Elites with the Iso+LineDD operator [249], and Multi-task MAP-Elites [170]. Unlike pymap_elites, pyribs offers a larger selection of algorithms under one framework.

7.2.3.4 QDax

QDax [150] is a recent library that was developed after the initial release of pyribs. The library focuses on efficient QD, reinforcement learning (RL), and evolutionary algorithm implementations for hardware accelerators such as GPUs and TPUs, taking advantage of the parallel nature of these methods. QDax specializes in reinforcement learning and robotics domains, where evaluation remains an expensive bottleneck. Many experiments that took hours or days on a CPU cluster take only minutes with GPU acceleration in QDax. To leverage accelerators in both function evaluation and algorithm implementation, QDax builds on the JAX library [25] and provides a JAX-based API.

While pyribs incorporates batch operations like those found in QDax to ensure a reasonable level of performance (Sec. 7.4.2.5), pyribs only runs single-threaded on a single CPU (Sec. 7.4.2.3). In addition, pyribs is not based on specialized libraries, which makes it accessible to a more general audience, such as beginners who have only basic Python knowledge and limited computational resources. Finally, while QDax extends beyond QD by providing baseline algorithms from RL and multi-objective optimization, pyribs focuses on general-purpose QD algorithms under the RIBS framework.

7.3 The RIBS Framework

Pyribs implements the conceptual RIBS framework that consists of three core components: (1) an *archive* storing solutions generated by the QD algorithm, (2) *emitters* generating solutions for evaluation, and (3) a *scheduler* managing the interaction of the archive and emitters and providing the primary ask-tell [109] interface to the user. Algorithm 11 shows the standard execution loop for combining these components. As we show in Sec. 7.3.2.2, this execution loop is flexible and not limited to a single call to the ask-tell interface.

7.3.1 Components

7.3.1.1 Archive

The archive is a data structure which stores solutions generated by the QD algorithm, along with any information relevant to solutions, such as objective and measure values. The primary archive method is `add()`, which takes in multiple solutions with their objective and measure values, attempts to add them to the collection of solutions, and

returns information about the addition. Examples of such information include "status" (whether the solution found a new cell in the archive, improved an existing cell, or was not added at all), "novelty" (the average distance in measure space from the solution to its k-nearest neighbors in the archive [145]), and "improvement value" (the difference between the solution's objective value and that of the solution which it replaced [72]). Archives may support additional functionality, such as methods for sampling solutions and retrieving solutions with given measure values.

An important choice in the implementation of `add()` is the order of inserting solutions. The simplest choice is to insert solutions *sequentially*, i.e., one after another. Pyribs offers sequential addition but defaults to the alternative of inserting all solutions simultaneously as a *batch*.

Batching has the following benefits. First, some metrics depend on the order in which solutions are inserted. For example, if two solutions $\boldsymbol{\theta}_a$ and $\boldsymbol{\theta}_b$ have similar measures, then $\boldsymbol{\theta}_a$ may be inserted with high novelty, while $\boldsymbol{\theta}_b$ is subsequently inserted with low novelty because $\boldsymbol{\theta}_a$ is already in the archive. Batching overcomes this issue by "freezing" the archive, then computing the metrics of all solutions with respect to the frozen archive. Second, batching enables enhanced performance, as libraries like NumPy (used in pyribs) and JAX (used in QDax) are designed to operate on batches of data.

7.3.1.2 Emitters

QD algorithms in RIBS instantiate one or more emitters. Emitters are algorithms that generate solutions and adapt to objective, measure, and archive insertion feedback. Emitters in RIBS provide two methods. The `ask()` method queries the emitter's algorithm for candidate solutions. The `tell()` method updates the internal algorithm state based on the objective and measure values of the generated solutions and any information gained from adding the solutions to the archive.

One example of a RIBS emitter is the CMA-ES emitter from CMA-ME [77]. Here, calling `ask()` samples solutions from the Gaussian distribution maintained by CMA-ES, while calling `tell()` updates the Gaussian distribution and internal CMA-ES parameters [108].

It is also possible that emitters in RIBS do not require any internal state. For instance, when `ask()` is called, one variation of MAP-Elites generates new solutions by sampling existing archive solutions and perturbing them with

fixed-variance Gaussian noise. Since there are no parameters to update for this Gaussian noise mutation, the `tell()` method does not perform any operation.

7.3.1.3 Scheduler

The scheduler performs two roles in the RIBS framework. First, the scheduler facilitates the interaction between the archive and the population of emitters. The scheduler adds solutions generated by emitters to the archive and passes the results of evaluation and archive insertion to the emitters. Second, schedulers select which emitters generate new solutions on each iteration of the algorithm. Schedulers make decisions on active emitters based on how well each emitter performs in previous iterations.

Schedulers implement an ask-tell interface as shown in Algorithm 11. When `ask()` is called (line 11), the scheduler selects one or more emitters and calls each emitter's `ask()` method to generate solutions. When `tell()` is called (line 18), the scheduler takes in the objective and measure function evaluations of these solutions and `add()`'s the solutions to the archive. Then, the scheduler passes the solutions, evaluations, and archive addition information to the emitters via each emitter's `tell()` method.

In the original emitter implementation [77], emitters directly called `add()` to insert solutions into the archive. However, allowing emitters to modify the archive meant that feedback from `add()` depended on the order in which emitters were called, similar to adding solutions sequentially in archives as discussed in Sec. 7.3.1.1. Now, although the emitters may read data from the archive (e.g., when sampling solutions), only the scheduler calls `add()` and passes the returned information to the emitters through their `tell()` method.

Ultimately, the scheduler provides the primary user interface in the RIBS framework. As shown in Algorithm 11, users directly call `ask()`, evaluate solutions, and pass the results to `tell()`.

7.3.2 Composing Algorithms in RIBS

Algorithm 11 shows a standard execution loop in RIBS. First, the user configures the core components. Then, in the main loop (line 6), the user calls the scheduler's ask-tell interface and evaluates solutions in between the calls. Importantly, the RIBS components (archive, emitters, and scheduler) in this loop are interchangeable, and the execution

Algorithm 11: Standard Execution Loop in RIBS

QD Algorithm (n_e, n_{it})**:**
  Input: Number of emitters n_e, number of iterations n_{it}, parameters for $Archive$, $Emitters$, and $Scheduler$
  Result: Generates solutions to optimize the QD objective, stored in an $Archive$
  $Archive \leftarrow$ init_archive()
  $[Emitter_1..Emitter_{n_e}] \leftarrow$ init_emitters($Archive$)
  $Scheduler \leftarrow$ init_scheduler($Archive$, $[Emitter_1..Emitter_{n_e}]$)
  for $itr \leftarrow 1..n_{it}$ **do**
    $L \leftarrow Scheduler$.ask()
    $User$ computes $Evals = [f(\boldsymbol{\theta}), \boldsymbol{m}(\boldsymbol{\theta})$ for $\boldsymbol{\theta}$ in $L]$
    $Scheduler$.tell($Evals$)
  return $Archive$
Scheduler.ask ()**:**
  Result: Returns a list of solutions L generated by the emitters.
  $L \leftarrow []$ `// Empty list`
  for $i \leftarrow 1..n_e$ **do**
    if $Emitter_i$ should generate solutions **then**
      $L_i \leftarrow Emitter_i$.ask()
      $L \leftarrow LL_i$ `// Concatenate` L_i `to` L
  return L
Scheduler.tell $(Evals)$**:**
  Input: Objective and measure function evaluations of the list of solutions L.
  Result: Inserts solutions into $Archive$ and updates $Emitters$.
  $add\_info \leftarrow Archive$.add($L, Evals$)
  for $i \leftarrow 1..n_e$ **do**
    if $Emitter_i$ generated solutions **then**
      Retrieve solutions L_i generated by $Emitter_i$
      Retrieve $Evals_i$ corresponding to L_i
      Retrieve add_info_i corresponding to L_i
      $Emitter_i$.tell($L_i, Evals_i, add\_info_i$)

loop can be customized to support new QD algorithms. We show how replacing components or modifying the execution loop enables RIBS to support a variety of QD algorithms.

7.3.2.1 Integrating Different Components

First, we consider algorithms which replace components of RIBS without modifying the standard execution loop of Algorithm 11. Table 7.1 summarizes the components required for each algorithm. Throughout this section, we italicize the components listed in Table 7.1 as we introduce them.

We begin with MAP-Elites [169], which has a *grid archive* that tessellates the measure space into a grid. MAP-Elites incorporates a single emitter that randomly selects solutions from the archive and applies mutations. One kind of mutation is to add Gaussian noise; in this case, we call the emitter the *Gaussian emitter*. As is common in many versions

of MAP-Elites, the Gaussian emitter can also sample directly in the solution space on initial calls to `ask()`. Since this emitter has no adaptive components, its `tell()` method does nothing. Finally, MAP-Elites has a *basic scheduler* that simply selects this emitter on every iteration.

Replacing components creates different MAP-Elites variants. Substituting the Gaussian emitter with the *Iso+LineDD emitter*, which applies the Iso+LineDD operation [249], results in Iso+LineDD MAP-Elites.[§] We can replace the archive with a *CVT archive* or *sliding boundaries archive* to obtain CVT-MAP-Elites [248] and MESB [78].

We can also consider methods based on Novelty Search like NSLC [146]. Here, the *unstructured archive* adds solutions if they are far away from their k nearest neighbors in the archive. Meanwhile, the *genetic algorithm emitter* contains a genetic algorithm such as NEAT [229] that manages a population of solutions. In contrast to the Gaussian and Iso+LineDD emitters, the genetic algorithm emitter's `tell()` method updates its internal population. The scheduler remains the same as in MAP-Elites.

CMA-ME [77] and CMA-MAE [72] are more complicated because they require managing multiple instances of CMA-ES in parallel. In this case, we create multiple *CMA-ES emitters*, each with their own CMA-ES instance. Calling `ask()` on each emitter samples solutions from CMA-ES's multivariate Gaussian distribution, and calling `tell()` updates the distribution parameters and the internal CMA-ES parameters. We combine these emitters with the grid archive and basic scheduler from MAP-Elites.

Multi-Emitter MAP-Elites (ME-MAP-Elites) [47] provides an example of an algorithm that requires a different scheduler. The default ME-MAP-Elites includes CMA-ES and Iso+LineDD emitters. Its *bandit scheduler* applies a multi-armed bandit algorithm to select a subset of these emitters based on whether they have previously generated solutions that were inserted into the archive.

7.3.2.2 Modifying the execution loop

Besides algorithms that replace components of RIBS, we also consider those that modify the RIBS execution loop. In this regard, CMA-MEGA [73] and CMA-MAEGA [72] both require a *gradient arborescence emitter*, which constructs solutions by branching from a solution point based on the objective and measure gradients. This branching requires calling `ask()` and `tell()` twice: once to collect the solution point and return its evaluations and gradients, and once

[§]The original Iso+LineDD MAP-Elites [249] uses a CVT archive, but the authors noted that a grid archive would also work with their algorithm.

Algorithm 12: QD Algorithm with Surrogate Model in RIBS

1 **QD Algorithm with Surrogate Model** $(n_e, n_{inner}, n_{outer})$**:**
  Input: Number of emitters n_e, inner loop iterations n_{inner}, outer loop iterations n_{outer}, parameters for $Archive$, $Emitters$, $Scheduler$, and $Model$
  Result: Generates solutions to optimize the QD objective, stored in an $Archive$
2 $Archive \leftarrow$ init_archive()
3 $Model \leftarrow$ init_surrogate_model()
4 $\mathcal{D} \leftarrow \{\}$ `// Dataset of evaluated solutions`
5 **for** $itr \leftarrow 1..n_{outer}$ **do**
6 `// Construct surrogate archive.`
7 $Archive' \leftarrow$ init_archive()
8 $[Emitter'_1..Emitter'_{n_e}] \leftarrow$ init_emitters($Archive'$)
9 $Scheduler' \leftarrow$ init_scheduler($Archive'$,
10 $[Emitter'_1..Emitter'_{n_e}]$)
11 **for** $iter \leftarrow 1..n_{inner}$ **do**
12 $L \leftarrow Scheduler'$.ask()
13 $Evals' \leftarrow [Model.f(\boldsymbol{\theta}), Model.\boldsymbol{m}(\boldsymbol{\theta})$ for $\boldsymbol{\theta}$ in $L]$
14 $Scheduler'$.tell($Evals'$)
15 `// Record true evaluations of solutions.`
16 $L \leftarrow$ all solutions in $Archive'$
17 $User$ computes $Evals = [f(\boldsymbol{\theta}), \boldsymbol{m}(\boldsymbol{\theta})$ for $\boldsymbol{\theta}$ in $L]$
18 $Archive$.add($L, Evals$)
19 `// Update model.`
20 $\mathcal{D} \leftarrow \mathcal{D} \cup (L, Evals)$
21 Train $Model$ on data in $\mathcal{D}$
22 **return** $Archive$

to handle the branched solutions. Compared to Algorithm 11, we add another set of calls to `ask()` and `tell()` in the loop on line 6, with appropriate arguments to handle passing gradients back to `tell()`.

In addition, a number of recent works [83, 267, 19, 130] integrate surrogate models with QD algorithms in domains where evaluations are expensive. Surrogate-assisted QD algorithms construct an archive based on evaluations predicted by a surrogate model and then select candidate solutions for ground-truth evaluations.

Algorithm 12 shows a general layout for such an algorithm. This algorithm maintains a *ground-truth archive* for storing solutions which have been evaluated by the user (line 2). Then, during an *outer loop* (line 5), it performs three phases. First, it constructs a *surrogate archive* in an *inner loop* (line 11) based on solutions evaluated by the model (line 13). Second, the user evaluates the candidate solutions from the surrogate archive, and the evaluated solutions are added into the ground-truth archive (line 18). Finally, the algorithm trains the model to improve its predictions (line 21).

7.4 Designing pyribs

To realize the RIBS framework, we created the pyribs library and released it in 2021. The structure of the library closely follows the framework, with subpackages for archives, emitters, and schedulers. We describe the principles that have guided our implementation decisions and notable features that highlight these principles.

7.4.1 Principles

7.4.1.1 Simple

We designed pyribs to be "bare-bones" and maintain only the core components required for a QD algorithm optimizing a continuous search space. The simplicity of the design makes the library easier for new users to adopt, while the focus on continuous optimization problems reduces implementation complexity and makes the defined search space less abstract.

7.4.1.2 Flexible

Pyribs is also "bare-bones" in the sense that the core components of the library — archives, emitters, and schedulers — are all exposed to the user. This allows users to easily exchange components of the QD algorithm, and the design provides a foundation to implement future QD algorithms discovered by researchers.

7.4.1.3 Accessible

Pyribs is accessible to a wide audience, ranging from beginners to experienced researchers, by having readable source code, being easy to install, and having full documentation defining its usage. Our dependency choices ensure that beginners with limited computational resources or basic hardware can install pyribs and study the tutorials. The library also supports experienced researchers by being amenable to modifications.

7.4.2 Implementation Features

These features demonstrate how our implementation choices align with the design principles of section 7.4.1.

7.4.2.1 Choice of Python

Python offers many desirable features. Beyond being a beginner-friendly language, it has a flourishing ecosystem, with package repositories like the Python Package Index (PyPI) [192] and Anaconda [8] providing easy access to many useful libraries. While Python itself is slower than lower-level languages like C++, libraries like NumPy [111] compensate for this limitation by providing access to efficient numerical computation routines. Python can also integrate with other programming languages through various packages; for instance, PyJNIus [138] enables running Python-based QD algorithms [75, 19] with the Mario AI Framework [133], a benchmark implemented in Java. Furthermore, Python has become the *de facto* language of machine learning, and with the influx of QD applications to machine learning [267, 73, 175, 75], it is important to support QD researchers from that area. Thus, implementing the RIBS framework in Python and distributing pyribs on PyPI and Anaconda makes pyribs *accessible*, as users can easily install and learn to use the library.

7.4.2.2 Focus on continuous optimization

To maintain *simplicity*, pyribs only supports continuous optimization problems with a fixed number of parameters. Such problems are ubiquitous in a variety of fields, including machine learning, and continuous fixed-length solutions are readily represented in software as arrays that can be efficiently manipulated by libraries like NumPy.

There are many other solution encodings that a QD library could support. For instance, discrete solutions (e.g., a list of integers) can be implemented with the same arrays used in pyribs, but recent research in QD has focused on continuous domains [35]. Alternatively, a QD library could support objects such as solutions of variable length, graphs [251], or neural network structure encodings from NEAT [229]. Although the RIBS framework is not limited to any one solution encoding, such structures are often domain-specific, and including them would increase the complexity of pyribs.

7.4.2.3 Single CPU

To be *simple* and *accessible*, pyribs runs single-threaded on a single CPU. Thus, pyribs can run on hardware ranging from laptops to high-performance clusters. While being single-threaded may limit the performance of pyribs, the runtime in many QD problems is dominated by the user's evaluation of solutions, rather than by the execution of the

QD algorithm in pyribs. Moreover, the need for high-performance algorithm implementations is already fulfilled by libraries like Sferes$_{v2}$ [167] and QDax [150]. However, if internal algorithm runtime grows to be a bottleneck in QD problems, we could redesign pyribs to optionally parallelize execution by putting each emitter on a separate thread. Note that while pyribs itself runs on a single CPU, evaluations are left to the user and may be parallelized as described in the next section.

7.4.2.4 Evaluations are left to the user

Since evaluations are often the bottleneck in QD, we considered providing utilities for running evaluations of solutions in parallel, as is done in QDpy [31]. We decided against doing so since many evaluation functions require specific dependencies and hardware configurations that are difficult to support in a general-purpose library. In short, we maintain *simplicity* by leaving evaluations to the user. Nevertheless, our documentation includes basic examples of how to integrate parallelism into pyribs workflows. For instance, our tutorial "Using CMA-ME to Land a Lunar Lander Like a Space Shuttle" parallelizes evaluations in only two lines of code with Python's `multiprocessing` module.

7.4.2.5 Batch operations

The initial version of pyribs implemented many operations sequentially, including the archives' `add()` methods (Sec. 7.3.1.1). To compensate for the reduced performance, we added the just-in-time compiler Numba [7] to many of these functions. However, multiple users indicated that doing so decreased *accessibility* by making the library difficult to modify and debug. Hence, in the most recent version of pyribs (0.5.0), we have re-implemented these methods as batch operations without Numba. For instance, instead of adding one solution to the archive at a time, pyribs now leverages NumPy array operations to add a batch of solutions simultaneously. These operations improve code readability and performance over the sequential implementations while still operating on a single CPU. For instance, on the 20D sphere linear projection benchmark, the runtime of CMA-ME decreased from 140s with sequential+Numba to 60s with batch implementations.

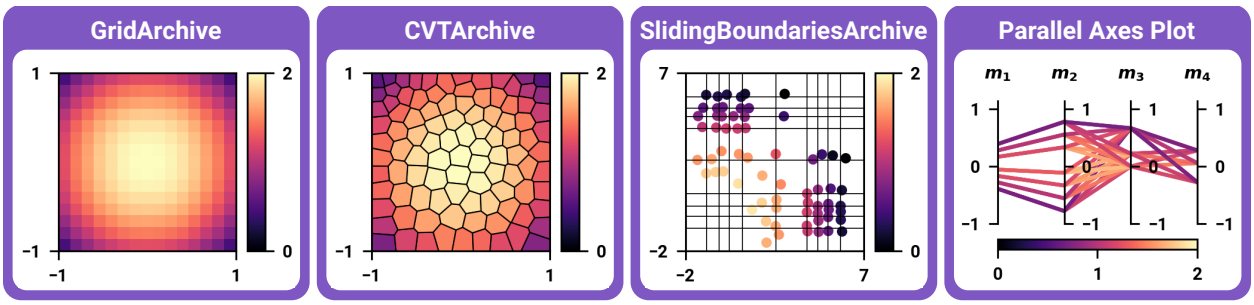


Figure 7.2: Pyribs visualization tools. We show example 2D heatmaps, where the axes correspond to the measure values, and the color of each archive cell indicates its objective value. In `SlidingBoundariesArchive`, the points show the locations of solutions in measure space, and the lines show the grid boundaries. We also show a *parallel axes plot* which can visualize an archive of any dimensionality. In this plot, a single solution's measures are plotted as a line connecting the measures $m_1 \ldots m_k$, and the line is colored according to the solution's objective value.

7.4.2.6 One-layer hierarchy

Ideally, every pyribs component would be implemented in a single file with no dependencies. Doing so would make the source code highly *accessible*, as a user could easily read the code for a component and modify it, similar to pymap_elites [168]. However, since components often share functionality, standalone files would lead to duplicate code and hamper maintenance. We compromise by introducing a one-layer hierarchy, where archives, emitters, and schedulers all inherit from their respective base classes. For instance, a user can learn about the implementation of `GridArchive` by reading the source code for `ArchiveBase` and `GridArchive`. This hierarchy helps make pyribs *flexible*, as users who create new components can inherit from these base classes instead of re-implementing basic functionality.

7.4.2.7 Visualization tools

Since there are no commonly available visualization tools for archives, we added our own to make pyribs *accessible*. As shown in Fig. 7.2, pyribs features heatmap visualizations for all its archives, as well as a parallel axes plot method that can visualize an archive of any dimensionality.

7.4.2.8 Documentation and tutorials

A key feature for increasing *accessibility* in pyribs is its extensive documentation and tutorials. Every pyribs component is documented in detail, and pyribs has an array of tutorials (Fig. 7.3). These tutorials teach users about the library and

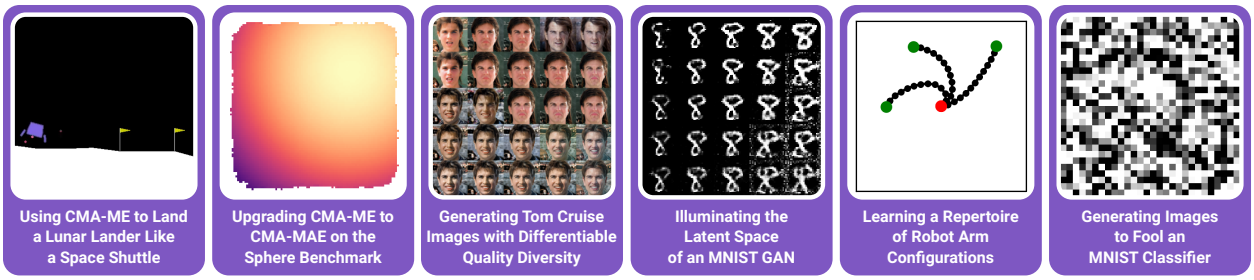


Figure 7.3: Tutorials enable pyribs users to quickly learn about the library and experiment with problems from the QD literature.

introduce them to common QD problems, such as latent space illumination [75], the arm repertoire benchmark [50, 249], and the sphere linear projection benchmark [77].

7.4.2.9 Industry standard practices

We draw from industry standard style guides [93] to promote readability and correctness in our source code. We automatically format our code with yapf [94] and check for basic errors with Pylint [191]. Furthermore, we implement a comprehensive suite of unit tests. Since it is difficult to test stochastic components like emitters and schedulers, our total code coverage by unit tests[¶] as of version 0.5.0 is 81%, but on archives, which are nearly deterministic, our coverage is 97%. Finally, when implementing new components, we run them on benchmarks such as sphere linear projection [77] to verify that our implementation matches results from prior work. These practices ensure that future changes in pyribs do not affect existing functionality.

7.5 Logo

Figure 7.4: The pyribs logo.

[¶]Code coverage measures the proportion of library code that is executed in tests.

Modern libraries such as PyTorch [183] have logos to help users recognize and identify the library in talks, tutorials, and other materials. Usually, the design of the logo is somehow connected to the design or name of the library.

In pyribs, we connect the design of the logo (Fig. 7.4) to the RIBS acronym, as well as to the dual-sense meaning of *bare-bones*. Recall that pyribs is bare-bones in two senses. First, pyribs maintains *simplicity* by only maintaining core components of a QD algorithm. Second, the library is *bare-bones* in the sense that the core components of the library — archives, emitters, and schedulers — are all exposed to the user.

We capture a similar spirit in the design of the pyribs logo by depicting ribs in two senses. The logo depicts a smoker, commonly used to cook beef or pork ribs, overlaid with a human rib cage. The smoker depicts the simplicity aspect of *bare-bones*, as pyribs is a useful tool because of its simplicity, while the rib cage captures the exposed nature of core pyribs components, allowing easy modification. The choice of the cooking-themed logo design is also closely connected to the modular nature of pyribs, allowing users to *cook up* new QD algorithms by exchanging components.

The pyribs logo lettering captures another dual-sense nature of pyribs. The RIBS framework is a conceptual framework for QD algorithms. The lettering of "ribs" augments a bone typeface to capture the abstract nature of the framework and draw a connection to the *bare-bones* conceptual design of pyribs. Meanwhile, the "py" is written in a Python-style typeface to connect with the Python implementation of pyribs.

Overall, we believe that a thoughtful and high-quality logo raises the cachet of pyribs and will help attract both researchers and practitioners to the library.

7.6 Conclusion

This chapter details the design of the conceptual RIBS framework and its implementation in the pyribs library. We show how RIBS supports a wide range of QD algorithms by interchanging core components, and we highlight the design principles of the library — simplicity, flexibility, and accessibility.

In the long run, our goal is for pyribs to become a library that supports a wide range of users in the QD community. On one end of the spectrum, we seek to support beginners by making pyribs ever more easy to learn and use. To this end, we will continue to maintain our documentation and tutorials and incorporate user feedback in our implementation decisions. Our vision is that pyribs will become an entry point into QD for many researchers.

On the other end of the spectrum, we aim to serve the needs of more experienced researchers by further developing the capabilities of pyribs. A major avenue in this direction is to expand the collection of pyribs components. Pyribs currently centers on the MAP-Elites family, but potential additions outside this family include the unstructured archive from Novelty Search, the archive with learned measures from AURORA [46], and emitters and schedulers from NS-ES [44] and SERENE [179].

Simultaneously, it is important for pyribs to support integrations with other fields. For example, many recent QD papers combine QD with deep learning [75, 73, 72, 175, 19]. Given the prevalence of GPUs in deep learning, it could thus be helpful to add GPU support to overcome delays incurred by transferring data between CPU and GPU. However, such advanced features will need to be delicately balanced against our design principles.

Beyond directly supporting practitioners, we believe that the lessons learned from the development of pyribs can inform the design and development of future QD libraries. We are thus excited about the supporting role that pyribs can play in expanding the QD community and growing QD into a widely adopted discipline.

Chapter 8

Conclusion

8.1 Summary

This dissertation proposed to make QD algorithms more accessible by scaling them up to address three key bottlenecks: (1) high solution space dimensionality, (2) long evaluation times, and (3) high measure space dimensionality. Addressing these limitations enhanced the applicability of QD algorithms in reinforcement learning, environment generation, and general machine learning, respectively. The final work in this dissertation proposed pyribs, a software package that encapsulated these advances and overall made QD more accessible to practitioners.

To address the first limitation and thus enable training neural networks for deep reinforcement learning, Chapters 2, 3, and 4 proposed methods that scale existing QD algorithms to high-dimensional solutions. Chapter 2 showed how to scale the existing CMA-MAE algorithm by introducing approximations of the CMA-ES covariance matrix. Thus, CMA-MAE could generate each solution in linear time rather than quadratic time, which enabled training neural network agents with tens of thousands of parameters. Next, while the scalable CMA-MAE variants in Chapter 2 were black-box algorithms that did not assume any information about the objective and measures, Chapter 3 built on differentiable quality diversity, which assumes the objective and measures are differentiable. Specifically, Chapter 3 proposed to approximate the gradients for DQD algorithms using reinforcement learning methods. Chapter 4 built on this idea by approximating the gradients with the on-policy reinforcement learning algorithm PPO, resulting in the state-of-the-art Proximal Policy Gradient Arborescence (PPGA) algorithm.

To address long evaluation times in environment generation, Chapter 5 integrated a surrogate model into the QD setup. This surrogate model synthesized information from prior evaluations (objectives, measures, and agent trajectories)

to approximate objectives and measures for new environments. By searching over this computationally cheap surrogate model, the Deep Surrogate Assisted Generation of Environments (DSAGE) algorithm created a set of solutions that were predicted to improve the archive of QD solutions. Since only solutions that were predicted to be valuable were evaluated, DSAGE improved sample efficiency over other QD algorithms.

To scale to higher measure space dimensionality, Chapter 6 introduced Discount Model Search (DMS). DMS addressed the issue of distortion, induced by many higher-dimensional measure spaces, by creating a smooth, continuous model of the discount function. Searching with this model enabled DMS to excel at exploring high-dimensional measure spaces. Furthermore, it enabled the Quality Diversity with Datasets of Measures (QDDM) setup, where desired measures are specified with a dataset of high-dimensional data. Given that datasets of high-dimensional data are ubiquitous in machine learning (e.g., datasets of images, videos, robot trajectories), we believe the QDDM setup greatly expands the range of machine learning problems that can be expressed as QD problems.

Chapter 7 introduced pyribs, a bare-bones Python library for QD. Pyribs is the software implementation of RIBS, a modular conceptual framework that includes three types of components: archives that store solutions, emitters that generate new solutions, and schedulers that orchestrate the QD algorithm. By combining these components together, pyribs can represent a wide range of QD algorithms, both from prior work in the QD community and from the methods discussed in this dissertation. Pyribs includes numerous well-maintained tutorials and extensive documentation, and it follows standard practices in the open source community for maintaining a healthy codebase. Since its inception in 2021, pyribs has powered many applications of QD in the community, and we envision it continuing to serve as a useful tool for practitioners.

Overall, these works combine to address bottlenecks at each stage of QD algorithms: generating solutions, evaluating solutions, and updating the archive. Tackling these limitations enables QD algorithms to expand their applications to previously unexplored areas.

8.2 Future Directions

This dissertation proposed enhancements to make QD algorithms more accessible to practitioners. In this section, I highlight some of the limitations of these advances and directions for future work.

8.2.1 Quality Diversity Reinforcement Learning

The methods for scaling to high-dimensional solutions in Chapters 2, 3, and 4 form a toolbox for quality diversity reinforcement learning (QD-RL). Each of these methods induces certain tradeoffs. For example, the scalable CMA-MAE variants are general black-box QD methods that can easily be applied to any problem, especially since they require very few hyperparameters. However, they do not leverage the structure of the RL problem, making it difficult to achieve maximal performance. On the other hand, PPGA deeply integrates with PPO and leverages the timestep-level information of the RL problem, enabling it to achieve state-of-the-art performance. However, it requires a large number of hyperparameters, making it challenging to tune for a new problem. In the future, an ideal method for QD-RL (and perhaps more broadly for high-dimensional solutions) would combine the best characteristics of the scalable CMA-MAE variants and PPGA: It would leverage information from the RL problem definition to achieve maximal performance, yet it would be easy to tune.

In the near future, such an ideal method may arise from advances in RL algorithms. Every year, new RL algorithms are developed with improvements such as higher performance or better ease-of-use. Since many QD-RL algorithms treat the RL algorithm as a modular component, it may be possible to improve one of the methods in this dissertation by replacing the RL algorithm with a newer one. For instance, robust policy optimization (RPO) [194] outperforms PPO in certain problems, so it may be a suitable replacement for PPO in PPGA. Already, prior work [151] has shown that the approach of replacing the RL algorithm with a new one can enhance other QD-RL algorithms, specifically PGA-ME [175].

In the longer term, I believe that developing an ideal method with few hyperparameters and high performance on benchmarks will gradually become less important. Instead, since RL is so popular, it seems more likely that a practitioner will already be familiar with an RL algorithm for their domain: they will know how to tune it on that domain, and they will have some hyperparameters that they know work well. Then, if they want to search for diverse solutions in their domain, they may turn to QD-RL methods. For this reason, I believe the focus will eventually shift to developing QD-RL algorithms and software that make it easy to "plug in" an existing RL algorithm in a modular fashion. Thus, when a practitioner decides to use a QD-RL method, they should be able to integrate the RL algorithm with which they are already familiar. In other words, this modular capability will lower the barrier to entry by enabling practitioners to leverage their existing intuitions.

Even further out, I foresee QD-RL being applied to many more problems, particularly extremely large ones. For example, fine-tuning of large language models (LLMs) currently comprises one of the most complex applications of RL algorithms [140]. In cases where diversely fine-tuned LLMs are required, e.g., to create different personalities for a chatbot, QD-RL could be applied to create an archive of LLMs. Doing so would likely require a QD-RL algorithm that can interface with the RL algorithms currently used to fine-tune LLMs. Ultimately, I believe that any area where an RL algorithm is currently applied could eventually be an area where QD-RL is applied. From this point of view, it is clear that developing QD-RL methods that integrate with any RL algorithm will make QD-RL accessible to a wide range of problems.

8.2.2 Large Language Models

One limitation of this dissertation's work on high-dimensional solutions is the assumption of a continuous solution space. Indeed, fixed-length continuous solution vectors are a powerful representation that occur in many machine learning problems. However, recent work has begun to integrate large language models (LLMs) with QD in order to apply QD in text-based domains like story writing [26]. One prominent application has been in *red-teaming* LLMs [206], i.e., searching for adversarial text prompts that induce undesired behaviors from existing LLMs. Instead of continuous solution vectors, the solutions in these applications are pieces of text, which are typically represented as "discrete" variable-length vectors of tokens. As such, methods like the scalable CMA-MAE variants are not directly applicable to these text-based domains. Nevertheless, given the recent popularity of LLMs, it may be fruitful to explore whether insights from continuous-solution QD algorithms can inform how QD algorithms use LLMs to search over text.

Meanwhile, surrogate models may be a useful tool for mitigating long evaluation times when working with LLMs in QD. Typically, LLMs are computationally expensive to run, so QD algorithms that involve them only run several thousand evaluations. Surrogate models that can predict the objective and measures for pieces of text may help accelerate the search. Such models would likely differ significantly from those designed for environment generation in Chapter 5, and they would require careful design to ensure they remain computationally cheap. On the other hand, leveraging foundation models as surrogate models may further expand the capabilities of QD algorithms. For instance, integrations with world models [11] could enable realistic surrogate models that facilitate direct applications of QD algorithms in the real world.

In the long term, it seems plausible that QD will become a common approach for testing LLMs. As the applications in red-teaming show, LLM testing is a suitable application for QD because it involves eliciting diverse behaviors of the LLM, especially harmful edge cases. The diversity here is well-represented as measure functions, and the set of behaviors is well-represented as an archive. Even QD algorithms that use continuous solutions may be applied here, as recent work trains "attacker" LLMs that each generate harmful prompts for a base LLM [255].

QD may also play a valuable role in creating diverse LLM-based agents. For example, recent work [227] showed how creating agents with diverse personalities enables simulating human behavior in collaboration environments. Lately, "Agentic AI" [3] has also emerged; it is an area where LLM-based agents orchestrate a wide variety of tasks. I believe there will be use cases that demand diverse agents, such as when personal preferences are involved. In such cases, it would be beneficial to fine-tune a diverse set of agents with QD and select from them to fulfill different users.

Finally, LLMs can help QD algorithms to become truly open-ended by determining what a QD algorithm should explore next. Currently, determining the diversity for a QD algorithm to explore is accomplished by defining measure functions. Using LLMs to quantify diversity and guide exploration may provide a path for automatically finding solutions that are "interesting" to users [266].

In the long run, it is likely that QD will continue to be integrated with LLMs and other foundation models. The principle of searching for diverse, high-performing solutions is a general one that can be applied in many problems, and foundation models play a crucial role in enabling this principle to be applied in complex problems.

8.2.3 Final Words

Overall, I believe the most important future direction for the QD community is to continue to search for challenges that are best formulated as QD problems. The methods presented in this dissertation and other prior works already form useful toolboxes. In many cases, they can be directly used without any modification, and they can even be combined in a modular fashion to solve problems that encounter more than one of the bottlenecks discussed in this dissertation. For example, prior works have applied QD to novel domains by running algorithms implemented in pyribs [166, 212]. I envision that continuing to search for such problems and enhance the accessibility of QD will grow QD into a standard, general-purpose approach in the practitioner's toolbox.

Bibliography

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. "TensorFlow: A System for Large-Scale Machine Learning". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI'16. Savannah, GA, USA: USENIX Association, 2016, pp. 265–283. ISBN: 9781931971331.

[2] Y. Abeysirigoonawardena, F. Shkurti, and G. Dudek. "Generating Adversarial Driving Scenarios in High-Fidelity Simulators". In: *Proceedings of the International Conference on Robotics and Automation (ICRA)*. 2019. DOI: 10.1109/ICRA.2019.8793740.

[3] Deepak Bhaskar Acharya, Karthigeyan Kuppan, and B. Divya. "Agentic AI: Autonomous Intelligence for Complex Goals—A Comprehensive Survey". In: *IEEE Access* 13 (2025), pp. 18912–18936. DOI: 10.1109/ACCESS.2025.3532853.

[4] Youhei Akimoto, Yuichi Nagata, Isao Ono, and Shigenobu Kobayashi. "Bidirectional Relation between CMA Evolution Strategies and Natural Evolution Strategies". In: *Parallel Problem Solving from Nature, PPSN XI*. Ed. by Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 154–163. ISBN: 978-3-642-15844-5.

[5] Maxime Allard. *Quality-Diversity Algorithms*. https://maximeallard.lu/2021/03/24/quality-diversity-algorithms/. Retrieved 2023-01-31. Mar. 2021.

[6] Shun-ichi Amari. "Natural Gradient Works Efficiently in Learning". In: *Neural Computation* 10.2 (Feb. 1998), pp. 251–276. ISSN: 0899-7667. DOI: 10.1162/089976698300017746. eprint: https://direct.mit.edu/neco/article-pdf/10/2/251/813415/089976698300017746.pdf.

[7] Anaconda Inc. *Numba: A Just-In-Time Compiler for Numerical Functions in Python*. https://numba.pydata.org. Computer software.

[8] Anaconda, Inc. *Anaconda*. https://anaconda.org.

[9] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf.

[10] James Arnold and Rob Alexander. “Testing Autonomous Robot Control Software Using Procedural Content Generation”. In: *Proceedings of the 32nd International Conference on Computer Safety, Reliability, and Security*. 2013. DOI: 10.1007/978-3-642-40793-2_4.

[11] Philip J. Ball, Jakob Bauer, Frank Belletti, Bethanie Brownfield, Ariel Ephrat, Shlomi Fruchter, Agrim Gupta, Kristian Holsheimer, Aleksander Holynski, Jiri Hron, Christos Kaplanis, Marjorie Limont, Matt McGill, Yanko Oliveira, Jack Parker-Holder, Frank Perbet, Guy Scully, Jeremy Shar, Stephen Spencer, Omer Tov, Ruben Villegas, Emma Wang, Jessica Yung, Cip Baetu, Jordi Berbel, David Bridson, Jake Bruce, Gavin Buttimore, Sarah Chakera, Bilva Chandra, Paul Collins, Alex Cullum, Bogdan Damoc, Vibha Dasagi, Maxime Gazeau, Charles Gbadamosi, Woohyun Han, Ed Hirst, Ashyana Kachra, Lucie Kerley, Kristian Kjems, Eva Knoepfel, Vika Koriakin, Jessica Lo, Cong Lu, Zeb Mehring, Alex Moufarek, Henna Nandwani, Valeria Oliveira, Fabio Pardo, Jane Park, Andrew Pierson, Ben Poole, Helen Ran, Tim Salimans, Manuel Sanchez, Igor Saprykin, Amy Shen, Sailesh Sidhwani, Duncan Smith, Joe Stanton, Hamish Tomlinson, Dimple Vijaykumar, Luyu Wang, Piers Wingfield, Nat Wong, Keyang Xu, Christopher Yew, Nick Young, Vadim Zubov, Douglas Eck, Dumitru Erhan, Koray Kavukcuoglu, Demis Hassabis, Zoubin Gharamani, Raia Hadsell, Aäron van den Oord, Inbar Mosseri, Adrian Bolton, Satinder Singh, and Tim Rocktäschel. “Genie 3: A New Frontier for World Models”. In: (2025).

[12] Thomas Bartz-Beielstein. “A survey of model-based methods for global optimization”. In: *Bioinspired Optimization Methods and Their Applications* (2016).

[13] Sumeet Batra, Zhehui Huang, Aleksei Petrenko, Tushar Kumar, Artem Molchanov, and Gaurav S. Sukhatme. “Decentralized Control of Quadrotor Swarms with End-to-end Deep Reinforcement Learning”. In: *Conference on Robot Learning, 8-11 November 2021, London, UK*. Ed. by Aleksandra Faust, David Hsu, and Gerhard Neumann. Vol. 164. Proceedings of Machine Learning Research. PMLR, 2021, pp. 576–586. URL: https://proceedings.mlr.press/v164/batra22a.html.

[14] Sumeet Batra, Bryon Tjanaka, Matthew C Fontaine, Aleksei Petrenko, Stefanos Nikolaidis, and Gaurav Sukhatme. “Proximal Policy Gradient Arborescence for Quality Diversity Reinforcement Learning”. In: *International Conference on Learning Representations* (2024).

[15] Robin Baumgarten. *Infinite Super Mario AI*. 2009. URL: https://wobblylabs.com/projects/marioai.

[16] Joachim Berg, Nils Gustav Andreas Berggren, Sivert Allergodt Borgeteien, Christian Ruben Alexander Jahren, Arqam Sajid, and Stefano Nichele. “Evolved art with transparent, overlapping, and geometric shapes”. In: *Symposium of the Norwegian AI Society*. Springer. 2019, pp. 3–15. URL: https://arxiv.org/abs/1904.06110.

[17] Hans-Georg Beyer and Hans-Paul Schwefel. “Evolution strategies: A comprehensive introduction”. In: *Natural Computing* 1.1 (Mar. 2002), pp. 3–52. ISSN: 1572-9796. DOI: 10.1023/A:1015059928466.

[18] Varun Bhatt, Heramb Nemlekar, Matthew Christopher Fontaine, Bryon Tjanaka, Hejia Zhang, Ya-Chuan Hsu, and Stefanos Nikolaidis. “Surrogate Assisted Generation of Human-Robot Interaction Scenarios”. In: *7th Annual Conference on Robot Learning*. 2023. URL: https://openreview.net/forum?id=C5MQUlzhVjQ.

[19] Varun Bhatt, Bryon Tjanaka, Matthew Fontaine, and Stefanos Nikolaidis. “Deep surrogate assisted generation of environments”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 37762–37777.

[20] Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. “A Tutorial on the Cross-Entropy Method”. In: *Annals of Operations Research* 134.1 (Feb. 2005), pp. 19–67. ISSN: 1572-9338. DOI: 10.1007/s10479-005-5724-z.

[21] Raphaël Boige, Guillaume Richard, Jérémie Dona, Thomas Pierrot, and Antoine Cully. "Gradient-Informed Quality Diversity for the Illumination of Discrete Spaces". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '23. Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 119–128. ISBN: 9798400701191. DOI: 10.1145/3583131.3590407.

[22] Philip Bontrager, Aditi Roy, Julian Togelius, Nasir Memon, and Arun Ross. "DeepMasterPrints: Generating MasterPrints for Dictionary Attacks via Latent Variable Evolution". In: *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS)*. 2018, pp. 1–9. DOI: 10.1109/BTAS.2018.8698539.

[23] David M Bossens and Danesh Tarapore. "QED: Using Quality-Environment-Diversity to Evolve Resilient Robot Swarms". In: *IEEE Transactions on Evolutionary Computation* (2020).

[24] David M. Bossens, Jean-Baptiste Mouret, and Danesh Tarapore. "Learning behaviour-performance maps with meta-evolution". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2020, pp. 49–57.

[25] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. *JAX: composable transformations of Python+NumPy programs*. http://github.com/jax-ml/jax. Version 0.3.13. 2018.

[26] Herbie Bradley, Andrew Dai, Hannah Benita Teufel, Jenny Zhang, Koen Oostermeijer, Marco Bellagente, Jeff Clune, Kenneth Stanley, Gregory Schott, and Joel Lehman. "Quality-Diversity through AI Feedback". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=owokKCrGYr.

[27] Dimo Brockhoff, Anne Auger, Nikolaus Hansen, Dirk V. Arnold, and Tim Hohm. "Mirrored Sampling and Sequential Selection for Evolution Strategies". In: *Parallel Problem Solving from Nature, PPSN XI*. Ed. by Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 11–21. ISBN: 978-3-642-15844-5.

[28] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "OpenAI Gym". In: *CoRR* abs/1606.01540 (2016). arXiv: 1606.01540. URL: http://arxiv.org/abs/1606.01540.

[29] Noam Brown and Tuomas Sandholm. "Superhuman AI for Multiplayer Poker". In: *Science* (2019).

[30] Kenny Cason. *Genetic Draw*. https://github.com/kennycason/genetic_draw. 2016. (Visited on 08/17/2021).

[31] L. Cazenille. *QDpy: A Python framework for Quality-Diversity*. https://gitlab.com/leo.cazenille/qdpy. 2018.

[32] Leo Cazenille, Nicolas Bredeche, and Nathanael Aubert-Kato. "Exploring self-assembling behaviors in a swarm of bio-micro-robots using surrogate-assisted map-elites". In: *IEEE Symposium Series on Computational Intelligence (SSCI)*. 2019.

[33] Felix Chalumeau, Raphael Boige, Bryan Lim, Valentin Macé, Maxime Allard, Arthur Flajolet, Antoine Cully, and Thomas Pierrot. "Neuroevolution is a Competitive Alternative to Reinforcement Learning for Skill Discovery". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=6BHlZgyPOZY.

[34] Allen Chang, Matthew C. Fontaine, Serena Booth, Maja J. Matarić, and Stefanos Nikolaidis. "Quality-diversity generative sampling for learning with synthetic data". In: *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. ISBN: 978-1-57735-887-9. DOI: 10.1609/aaai.v38i18.29955.

[35] Konstantinos Chatzilygeroudis, Antoine Cully, Vassilis Vassiliades, and Jean-Baptiste Mouret. "Quality-Diversity Optimization: A Novel Branch of Stochastic Optimization". In: *Black Box Optimization, Machine Learning, and No-Free Lunch Theorems*. Ed. by Panos M. Pardalos, Varvara Rasskazova, and Michael N. Vrahatis. Cham: Springer International Publishing, 2021, pp. 109–135. ISBN: 978-3-030-66515-9. DOI: 10.1007/978-3-030-66515-9_4.

[36] Yen-Chi Chen. "A tutorial on kernel density estimation and recent advances". In: *Biostatistics & Epidemiology* 1.1 (2017), pp. 161–187.

[37] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. *Minimalistic Gridworld Environment for OpenAI Gym*. https://github.com/maximecb/gym-minigrid. 2018.

[38] Tae Jong Choi and Julian Togelius. "Self-referential quality diversity through differential MAP-Elites". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2021. DOI: 10.1145/3449639.3459383.

[39] Jack Clark and Dario Amodei. *Faulty Reward Functions in the Wild*. https://openai.com/blog/faulty-reward-functions/. 2016.

[40] Jeff Clune, Joel Lehman, and Kenneth O. Stanley. *Recent Advances in Population-Based Search for Deep Neural Networks*. ICML 2019 Tutorials, https://youtu.be/g6HiuEnbwJE. 2019.

[41] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. "Leveraging procedural generation to benchmark reinforcement learning". In: *Proceedings of the International Conference on Machine Learning*. 2020.

[42] Cédric Colas, Vashisht Madhavan, Joost Huizinga, and Jeff Clune. "Scaling MAP-Elites to Deep Neuroevolution". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. GECCO '20. Cancún, Mexico: Association for Computing Machinery, 2020, pp. 67–75. ISBN: 9781450371285. DOI: 10.1145/3377930.3390217.

[43] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. "GEP-PG: Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 1039–1048. URL: https://proceedings.mlr.press/v80/colas18a.html.

[44] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth Stanley, and Jeff Clune. "Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., 2018, pp. 5027–5038. URL: http://papers.nips.cc/paper/7750-improving-exploration-in-evolution-strategies-for-deep-reinforcement-learning-via-a-population-of-novelty-seeking-agents.pdf.

[45] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org. 2016–2020.

[46] Antoine Cully. “Autonomous skill discovery with quality-diversity and unsupervised descriptors”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO ’19. Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 81–89. ISBN: 9781450361118. DOI: 10.1145/3321707.3321804.

[47] Antoine Cully. “Multi-emitter MAP-elites”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, June 2021. DOI: 10.1145/3449639.3459326.

[48] Antoine Cully. *Quality-Diversity optimisation algorithms*. https://quality-diversity.github.io. Retrieved 2023-01-31.

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

[50] Antoine Cully and Yiannis Demiris. “Quality and Diversity Optimization: A Unifying Modular Framework”. In: *IEEE Transactions on Evolutionary Computation* 22.2 (2018), pp. 245–259. DOI: 10.1109/TEVC.2017.2704781.

[51] Antoine Cully, Jean-Baptiste Mouret, and Stéphane Doncieux. “Quality-Diversity Optimisation”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. GECCO ’20. Cancún, Mexico: Association for Computing Machinery, 2020, pp. 701–723. ISBN: 9781450371278. DOI: 10.1145/3377929.3389852.

[52] Antoine Cully, Jean-Baptiste Mouret, and Stéphane Doncieux. “Quality-Diversity Optimisation”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO ’21. Lille, France: Association for Computing Machinery, 2021, pp. 715–739. ISBN: 9781450383516. DOI: 10.1145/3449726.3461403.

[53] Antoine Cully, Jean-Baptiste Mouret, and Stéphane Doncieux. “Quality-Diversity Optimisation”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO ’22. Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 864–889. ISBN: 9781450392686. DOI: 10.1145/3520304.3533637.

[54] K.A. De Jong. *Evolutionary Computation: A Unified Approach*. Bradford Books, 2006. ISBN: 9780262041942.

[55] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre M. Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. “Emergent Complexity and Zero-shot Transfer via Unsupervised Environment Design”. In: *Advances in Neural Information Processing Systems 33*. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/985e9a46e10005356bbaf194249f6856-Abstract.html.

[56] Aaron Dharna, Amy K Hoover, Julian Togelius, and Lisa Soros. “Transfer Dynamics in Emergent Evolutionary Curricula”. In: *IEEE Transactions on Games* (2022).

[57] Aaron Dharna, Julian Togelius, and Lisa B Soros. “Co-generation of game levels and game-playing agents”. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. 2020.

[58] Li Ding, Jenny Zhang, Jeff Clune, Lee Spector, and Joel Lehman. “Quality diversity through human feedback: towards open-ended diversity-driven optimization”. In: *Proceedings of the 41st International Conference on Machine Learning*. ICML’24. Vienna, Austria: JMLR.org, 2024.

[59] Qiang Du, Vance Faber, and Max Gunzburger. “Centroidal Voronoi Tessellations: Applications and Algorithms”. In: *SIAM Review* 41.4 (1999), pp. 637–676. DOI: 10.1137/S0036144599352836. eprint: https://doi.org/10.1137/S0036144599352836.

[60] Sam Earle, Maria Edwards, Ahmed Khalifa, Philip Bontrager, and Julian Togelius. “Learning controllable content generators”. In: *Proceedings of the IEEE Conference on Games (CoG)*. 2021.

[61] Sam Earle, Justin Snider, Matthew C Fontaine, Stefanos Nikolaidis, and Julian Togelius. “Illuminating diverse neural cellular automata for level generation”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2022, pp. 68–76.

[62] Adrien Ecoffet, Jeff Clune, and Joel Lehman. “Open Questions in Creating Safe Open-ended AI: Tensions Between Control and Creativity”. In: *CoRR* abs/2006.07495 (2020). URL: https://arxiv.org/abs/2006.07495.

[63] Benjamin Ellenberger. *PyBullet Gymperium*. https://github.com/benelot/pybullet-gym. 2018–2019.

[64] Teruto Endo, Hirotake Abe, and Mizuki Oka. “Toward automatic generation of diverse congestion control algorithms through co-evolution with simulation environments”. In: *ALIFE 2022: The 2022 Conference on Artificial Life*. July 2022. DOI: 10.1162/isal_a_00515. eprint: https://direct.mit.edu/isal/proceedings-pdf/isal/34/33/2035325/isal_a_00515.pdf.

[65] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. “Diversity is All You Need: Learning Skills without a Reward Function”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=SJx63jRqFm.

[66] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. “Testing the manifold hypothesis”. English (US). In: *Journal of the American Mathematical Society* 29.4 (Oct. 2016). Publisher Copyright: © 2016 American Mathmatical Society., pp. 983–1049. ISSN: 0894-0347. DOI: 10.1090/jams/852.

[67] Manon Flageat, Félix Chalumeau, and Antoine Cully. “Empirical Analysis of PGA-MAP-Elites for Neuroevolution in Uncertain Domains”. In: *ACM Trans. Evol. Learn. Optim.* 3.1 (Mar. 2023). ISSN: 2688-299X. DOI: 10.1145/3577203.

[68] Manon Flageat and Antoine Cully. “Fast and stable MAP-Elites in noisy domains using deep grids”. In: vol. ALIFE 2020: The 2020 Conference on Artificial Life. July 2020, pp. 273–282. DOI: 10.1162/isal_a_00316. eprint: https://direct.mit.edu/isal/proceedings-pdf/isal2020/32/273/1908606/isal_a_00316.pdf.

[69] Manon Flageat and Bryan Lim. *Benchmarking quality-diversity algorithms on neuroevolution for reinforcement learning*. https://aihub.org/2022/12/14/benchmarking-quality-diversity-algorithms-on-neuroevolution-for-reinforcement-learning/. Retrieved 2023-01-31. Dec. 2022.

[70] Michael Fogleman. *Primitive Pictures*. https://github.com/fogleman/primitive. 2016. (Visited on 08/17/2021).

[71] Matthew Fontaine and Stefanos Nikolaidis. “A quality diversity approach to automatically generating human-robot interaction scenarios in shared autonomy”. In: *Proceedings of Robotics: Science and Systems 17* (2021).

[72] Matthew Fontaine and Stefanos Nikolaidis. “Covariance matrix adaptation map-annealing”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2023, pp. 456–465.

[73] Matthew Fontaine and Stefanos Nikolaidis. “Differentiable Quality Diversity”. In: *Advances in Neural Information Processing Systems* 34 (2021).

[74] Matthew C Fontaine, Ya-Chuan Hsu, Yulun Zhang, Bryon Tjanaka, and Stefanos Nikolaidis. “On the importance of environments in human-robot coordination”. In: *Proceedings of Robotics: Science and Systems 17* (2021).

[75] Matthew C Fontaine, Ruilin Liu, Ahmed Khalifa, Jignesh Modi, Julian Togelius, Amy K Hoover, and Stefanos Nikolaidis. “Illuminating mario scenes in the latent space of a generative adversarial network”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2021, pp. 5922–5930.

[76] Matthew C Fontaine and Stefanos Nikolaidis. “Evaluating human–robot interaction algorithms in shared autonomy via quality diversity scenario generation”. In: *ACM Transactions on Human-Robot Interaction (THRI)* 11.3 (2022), pp. 1–30.

[77] Matthew C Fontaine, Julian Togelius, Stefanos Nikolaidis, and Amy K Hoover. “Covariance matrix adaptation for the rapid illumination of behavior space”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2020, pp. 94–102.

[78] Matthew C. Fontaine, Scott Lee, L. B. Soros, Fernando De Mesentier Silva, Julian Togelius, and Amy K. Hoover. “Mapping hearthstone deck spaces through MAP-elites with sliding boundaries”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO ’19. Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 161–169. ISBN: 9781450361118. DOI: 10.1145/3321707.3321794.

[79] Kevin Frans. *Quality Diversity: Evolving Ocean Creatures*. https://kvfrans.com/quality-diversity-evolving-ocean-creatures/. Retrieved 2023-01-31. Dec. 2020.

[80] Daniel J Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L Sangiovanni-Vincentelli, and Sanjit A Seshia. “Scenic: a language for scenario specification and scene generation”. In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2019.

[81] Scott Fujimoto, Herke van Hoof, and David Meger. “Addressing Function Approximation Error in Actor-Critic Methods”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. proceedings of machine learning research. pmlr, Oct. 2018, pp. 1587–1596. URL: http://proceedings.mlr.press/v80/fujimoto18a.html.

[82] Thomas Gabor, Andreas Sedlmeier, Marie Kiermeier, Thomy Phan, Marcel Henrich, Monika Pichlmair, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner SchmidSiemens AG, et al. “Scenario co-evolution for reinforcement learning on a grid world smart factory domain”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2019.

[83] Adam Gaier, Alexander Asteroth, and Jean-Baptiste Mouret. “Data-efficient design exploration through surrogate-assisted illumination”. In: *Evolutionary computation* 26.3 (2018), pp. 381–410.

[84] Adam Gaier, Alexander Asteroth, and Jean-Baptiste Mouret. “Discovering Representations for Black-Box Optimization”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. GECCO ’20. Cancún, Mexico: Association for Computing Machinery, 2020, pp. 103–111. ISBN: 9781450371285. DOI: 10.1145/3377930.3390221.

[85] Adam Gaier, James Stoddart, Lorenzo Villaggi, and Peter J. Bentley. “T-DominO”. In: *Parallel Problem Solving from Nature – PPSN XVII*. Ed. by Günter Rudolph, Anna V. Kononova, Hernán Aguirre, Pascal Kerschke, Gabriela Ochoa, and Tea Tušar. Cham: Springer International Publishing, 2022, pp. 263–277. ISBN: 978-3-031-14721-0.

[86] Theodoros Galanos, Antonios Liapis, Georgios N. Yannakakis, and Reinhard Koenig. “ARCH-Elites: Quality-Diversity for Urban Design”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO ’21. Lille, France: Association for Computing Machinery, 2021, pp. 313–314. ISBN: 9781450383516. DOI: 10.1145/3449726.3459490.

[87] Alessio Gambi, Marc Mueller, and Gordon Fraser. “Automatically Testing Self-driving Cars with Search-based Procedural Content Generation”. In: *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2019. DOI: 10.1145/3293882.3330566.

[88] Bruno Gašperov, Stjepan Begušić, Tessa Bauman, and Zvonko Kostanjčar. “Quality-diversity and Novelty Search for Portfolio Optimization and Beyond”. In: *Computational Economics* (May 2025). ISSN: 1572-9974. DOI: 10.1007/s10614-025-10985-2.

[89] Edoardo Giacomello, Pier Luca Lanzi, and Daniele Loiacono. “DOOM level generation using generative adversarial networks”. In: *Proceedings of the IEEE Games, Entertainment, Media Conference (GEM)*. 2018.

[90] Tobias Glasmachers, Tom Schaul, Sun Yi, Daan Wierstra, and Jürgen Schmidhuber. “Exponential natural evolution strategies”. In: *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. 2010, pp. 393–400.

[91] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, May 2010, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[92] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*. 2014.

[93] Google. *Google Python style Guide*. https://google.github.io/styleguide/pyguide.html. Retrieved 2023-02-09.

[94] Google. *YAPF: A formatter for Python files*. https://github.com/google/yapf. Computer software.

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

[96] Daniele Gravina, Antonios Liapis, and Georgios Yannakakis. “Surprise Search: Beyond Objectives and Novelty”. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. GECCO ’16. Denver, Colorado, USA: Association for Computing Machinery, 2016, pp. 677–684. ISBN: 9781450342063. DOI: 10.1145/2908812.2908817.

[97] Daniele Gravina, Antonios Liapis, and Georgios N. Yannakakis. “Quality Diversity Through Surprise”. In: *IEEE Transactions on Evolutionary Computation* 23.4 (2019), pp. 603–616. DOI: 10.1109/TEVC.2018.2877215.

[98] Luca Grillotti and Antoine Cully. “Kheperax: A Lightweight JAX-Based Robot Control Environment for Benchmarking Quality-Diversity Algorithms”. In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. GECCO ’23 Companion. Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 2163–2165. ISBN: 9798400701207. DOI: 10.1145/3583133.3596387.

[99] Luca Grillotti and Antoine Cully. "Relevance-guided unsupervised discovery of abilities with quality-diversity algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '22. Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 77–85. ISBN: 9781450392372. DOI: 10.1145/3512290.3528837.

[100] Luca Grillotti, Maxence Faldor, Borja González León, and Antoine Cully. "Quality-diversity actor-critic: learning high-performing and diverse behaviors via value and successor features critics". In: *Proceedings of the 41st International Conference on Machine Learning*. ICML'24. Vienna, Austria: JMLR.org, 2024.

[101] Matthew Guzdial and Mark Riedl. "Game level generation from gameplay videos". In: *Proceedings of the 12th Artificial Intelligence and Interactive Digital Entertainment Conference*. 2016.

[102] David Ha. "A Visual Guide to Evolution Strategies". In: *blog.otoro.net* (2017). URL: https://blog.otoro.net/2017/10/29/visual-evolution-strategies/.

[103] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 1861–1870. URL: https://proceedings.mlr.press/v80/haarnoja18b.html.

[104] Alexander Hagg, Sebastian Berns, Alexander Asteroth, Simon Colton, and Thomas Bäck. "Expressivity of parameterized and data-driven representations in quality diversity search". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2021, pp. 678–686.

[105] Alexander Hagg, Dominik Wilde, Alexander Asteroth, and Thomas Bäck. "Designing Air Flow with Surrogate-Assisted Phenotypic Niching". In: *Parallel Problem Solving from Nature – PPSN XVI*. Ed. by Thomas Bäck, Mike Preuss, André Deutz, Hao Wang, Carola Doerr, Michael Emmerich, and Heike Trautmann. Cham: Springer International Publishing, 2020, pp. 140–153. ISBN: 978-3-030-58112-1.

[106] Eric Hambro, Sharada P. Mohanty, Dmitrii Babaev, Minwoo Byeon, Dipam Chakraborty, Edward Grefenstette, Minqi Jiang, et al. "Insights From the NeurIPS 2021 NetHack Challenge". In: *CoRR* abs/2203.11889 (2022). DOI: 10.48550/arXiv.2203.11889.

[107] Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar Sundaralingam, Yashraj Narang, Jean-Francois Lafleche, Dieter Fox, and Gavriel State. "DeXtreme: Transfer of Agile In-Hand Manipulation from Simulation to Reality". In: *arXiv* (2022).

[108] Nikolaus Hansen. "The CMA Evolution Strategy: A Tutorial". In: *arXiv preprint arXiv:1604.00772* (2016).

[109] Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. *CMA-ES/pycma on Github*. Zenodo, DOI:10.5281/zenodo.2559634. Feb. 2019. DOI: 10.5281/zenodo.2559634.

[110] Nikolaus Hansen and Andreas Ostermeier. "Completely Derandomized Self-Adaptation in Evolution Strategies". In: *Evolutionary Computation* 9.2 (June 2001), pp. 159–195. ISSN: 1063-6560. DOI: 10.1162/106365601750190398. eprint: https://direct.mit.edu/evco/article-pdf/9/2/159/1493523/106365601750190398.pdf.

[111] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[112] Horace He. *The State of Machine Learning Frameworks in 2019*. https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/. 2019.

[113] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[114] Shashank Hegde, Sumeet Batra, K.R. Zentner, and Gaurav S. Sukhatme. "Generating Behaviorally Diverse Policies with Latent Diffusion Models". In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: https://openreview.net/forum?id=nafgeYknRT.

[115] Dan Hendrycks, Nicholas Carlini, John Schulman, and Jacob Steinhardt. "Unsolved Problems in ML Safety". In: *CoRR* abs/2109.13916 (2021). URL: https://arxiv.org/abs/2109.13916.

[116] Eugenio Herrera-Berg. *StyleGAN3 CLIP-based guidance*. https://github.com/ouhenio/StyleGAN3-CLIP-notebooks. 2021.

[117] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. "CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms". In: *Journal of Machine Learning Research* 23.274 (2022), pp. 1–18. URL: http://jmlr.org/papers/v23/21-1342.html.

[118] Johann Huber, François Helenon, Miranda Coninx, Faïz Ben Amar, and Stéphane Doncieux. "Quality Diversity under Sparse Interaction and Sparse Reward: Application to Grasping in Robotics". In: *Evolutionary Computation* (Jan. 2025), pp. 1–30. ISSN: 1063-6560. DOI: 10.1162/evco_a_00363. eprint: https://direct.mit.edu/evco/article-pdf/doi/10.1162/evco_a_00363/2497988/evco_a_00363.pdf.

[119] Institute of Digital Games. *Game AI - Creative artificial evolution through quality diversity algorithms*. https://www.game.edu.mt/blog/game-ai-creative-artificial-evolution-through-quality-diversity-algorithms/. Retrieved 2023-01-31. Apr. 2019.

[120] Alex Irpan. *Deep Reinforcement Learning Doesn't Work Yet*. https://www.alexirpan.com/2018/02/14/rl-hard.html. 2018.

[121] Nick Jakobi. "Evolutionary Robotics and the Radical Envelope-of-Noise Hypothesis". In: *Adaptive Behavior* (1998). DOI: 10.1177/105971239700600205.

[122] Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob N. Foerster, Edward Grefenstette, and Tim Rocktäschel. "Replay-Guided Adversarial Environment Design". In: *Advances in Neural Information Processing Systems 34*. 2021. URL: https://proceedings.neurips.cc/paper/2021/hash/0e915db6326b6fb6a3c56546980a8c93-Abstract.html.

[123] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. "Prioritized Level Replay". In: *Proceedings of the 38th International Conference on Machine Learning, ICML*. 2021. URL: http://proceedings.mlr.press/v139/jiang21b.html.

[124] Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. "Illuminating generalization in deep reinforcement learning through procedural level generation". In: *arXiv preprint arXiv:1806.10729* (2018).

[125] Sergey Karakovskiy and Julian Togelius. "The Mario AI Benchmark and Competitions". In: *IEEE Transactions on Computational Intelligence and AI in Games* (2012). DOI: 10.1109/TCIAIG.2012.2188528.

[126] Daniel Karavolos, Antonios Liapis, and Georgios N. Yannakakis. "A Multifaceted Surrogate Model for Search-Based Procedural Content Generation". In: *IEEE Transactions on Games* (2021).

[127] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. "Alias-free generative adversarial networks". In: *Proceedings of the 35th International Conference on Neural Information Processing Systems*. NIPS '21. Red Hook, NY, USA: Curran Associates Inc., 2021. ISBN: 9781713845393.

[128] Tero Karras, Samuli Laine, and Timo Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.12 (2021), pp. 4217–4228. DOI: 10.1109/TPAMI.2020.2970919.

[129] Leon Keller, Daniel Tanneberg, Svenja Stark, and Jan Peters. "Model-Based Quality-Diversity Search for Efficient Robot Learning". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. 2020. DOI: 10.1109/IROS45743.2020.9340794.

[130] Paul Kent, Adam Gaier, Jean-Baptiste Mouret, and Juergen Branke. "Bayesian Optimization for Quality Diversity Search With Coupled Descriptor Functions". In: *IEEE Transactions on Evolutionary Computation* 29.2 (2025), pp. 302–316. DOI: 10.1109/TEVC.2024.3376733.

[131] Shauharda Khadka, Somdeb Majumdar, Tarek Nassar, Zach Dwiel, Evren Tumer, Santiago Miret, Yinyin Liu, and Kagan Tumer. "Collaborative Evolutionary Reinforcement Learning". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 3341–3350. URL: https://proceedings.mlr.press/v97/khadka19a.html.

[132] Shauharda Khadka and Kagan Tumer. "Evolution-Guided Policy Gradient in Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/85fc37b18c57097425b52fc7afbb6969-Paper.pdf.

[133] Ahmed Khalifa. *Mario AI Framework*. https://github.com/amidos2006/Mario-AI-Framework. 2019.

[134] Ahmed Khalifa, Philip Bontrager, Sam Earle, and Julian Togelius. "PCGRL: Procedural Content Generation via Reinforcement Learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. 2020.

[135] Ahmed Khalifa, Scott Lee, Andy Nealen, and Julian Togelius. "Talakat: Bullet hell generation through constrained map-elites". In: *Proceedings of The Genetic and Evolutionary Computation Conference*. 2018, pp. 1047–1054.

[136] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[137] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. "A Survey of Generalisation in Deep Reinforcement Learning". In: *CoRR* abs/2111.09794 (2021). URL: https://arxiv.org/abs/2111.09794.

[138] Kivy Team et al. *PyJNIus*. https://github.com/kivy/pyjnius. 2010.

[139] Saurabh Kumar, Aviral Kumar, Sergey Levine, and Chelsea Finn. "One Solution is Not All You Need: Few-Shot Extrapolation via Structured MaxEnt RL". In: *Advances in Neural Information Processing Systems* 33 (2020).

[140] Nathan Lambert. *Reinforcement Learning from Human Feedback*. Online, 2025. URL: https://rlhfbook.com.

[141] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[142] David H. Lee, Anishalakshmi Palaparthi, Matthew C. Fontaine, Bryon Tjanaka, and Stefanos Nikolaidis. "Density Descent for Diversity Optimization". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2024, pp. 674–682.

[143] Joel Lehman, Jay Chen, Jeff Clune, and Kenneth O. Stanley. "ES is More than Just a Traditional Finite-Difference Approximator". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '18. Kyoto, Japan: Association for Computing Machinery, 2018, pp. 450–457. ISBN: 9781450356183. DOI: 10.1145/3205455.3205474.

[144] Joel Lehman, Jeff Clune, Dusan Misevic, Christoph Adami, Lee Altenberg, Julie Beaulieu, Peter J. Bentley, Samuel Bernard, Guillaume Beslon, David M. Bryson, Nick Cheney, Patryk Chrabaszcz, Antoine Cully, Stephane Doncieux, Fred C. Dyer, Kai Olav Ellefsen, Robert Feldt, Stephan Fischer, Stephanie Forrest, Antoine Fŕenoy, Christian Gagńe, Leni Le Goff, Laura M. Grabowski, Babak Hodjat, Frank Hutter, Laurent Keller, Carole Knibbe, Peter Krcah, Richard E. Lenski, Hod Lipson, Robert MacCurdy, Carlos Maestre, Risto Miikkulainen, Sara Mitri, David E. Moriarty, Jean-Baptiste Mouret, Anh Nguyen, Charles Ofria, Marc Parizeau, David Parsons, Robert T. Pennock, William F. Punch, Thomas S. Ray, Marc Schoenauer, Eric Schulte, Karl Sims, Kenneth O. Stanley, François Taddei, Danesh Tarapore, Simon Thibault, Richard Watson, Westley Weimer, and Jason Yosinski. "The Surprising Creativity of Digital Evolution: A Collection of Anecdotes from the Evolutionary Computation and Artificial Life Research Communities". In: *Artificial Life* 26.2 (May 2020), pp. 274–306. ISSN: 1064-5462. DOI: 10.1162/artl_a_00319. eprint: https://direct.mit.edu/artl/article-pdf/26/2/274/1896071/artl_a_00319.pdf.

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

[146] Joel Lehman and Kenneth O. Stanley. "Evolving a diversity of virtual creatures through novelty search and local competition". In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. GECCO '11. Dublin, Ireland: Association for Computing Machinery, 2011, pp. 211–218. ISBN: 9781450305570. DOI: 10.1145/2001576.2001606.

[147] Yunzhu Li, Jiaming Song, and Stefano Ermon. "InfoGAIL: Interpretable Imitation Learning from Visual Demonstrations". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/2cd4e8a2ce081c3d7c32c3cde4312ef7-Paper.pdf.

[148] Diego Perez Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, and Simon M. Lucas. "General Video Game AI: Competition, Challenges and Opportunities". In: *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. 2016. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11853.

[149] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: http://arxiv.org/abs/1509.02971.

[150] Bryan Lim, Maxime Allard, Luca Grillotti, and Antoine Cully. "Accelerated Quality-Diversity for Robotics through Massive Parallelism". In: *arXiv preprint arXiv:2202.01258* (2022).

[151] Bryan Lim, Manon Flageat, and Antoine Cully. "Understanding the Synergies between Quality-Diversity and Deep Reinforcement Learning". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '23. Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 1212–1220. ISBN: 9798400701191. DOI: 10.1145/3583131.3590388.

[152] Bryan Lim, Luca Grillotti, Lorenzo Bernasconi, and Antoine Cully. "Dynamics-Aware Quality-Diversity for Efficient Learning of Skill Repertoires". In: *2022 International Conference on Robotics and Automation (ICRA)*. Philadelphia, PA, USA: IEEE Press, 2022, pp. 5360–5366. DOI: 10.1109/ICRA46639.2022.9811559.

[153] Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N Yannakakis, and Julian Togelius. "Deep learning for procedural content generation". In: *Neural Computing and Applications* (2021).

[154] I. Loshchilov, T. Glasmachers, and H. Beyer. "Large Scale Black-Box Optimization by Limited-Memory Matrix Adaptation". In: *IEEE Transactions on Evolutionary Computation* 23.2 (2019), pp. 353–358. DOI: 10.1109/TEVC.2018.2855049.

[155] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. "Isaac Gym: High Performance GPU Based Physics Simulation For Robot Learning". In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*. Ed. by Joaquin Vanschoren and Sai-Kit Yeung. 2021. URL: https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/28dd2c7955ce926456240b2ff0100bde-Abstract-round2.html.

[156] Vongani H. Maluleke, Neerja Thakkar, Tim Brooks, Ethan Weber, Trevor Darrell, Alexei A. Efros, Angjoo Kanazawa, and Devin Guillory. "Studying Bias in GANs Through the Lens of Race". In: *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XIII*. Tel Aviv, Israel: Springer-Verlag, 2022, pp. 344–360. ISBN: 978-3-031-19777-2. DOI: 10.1007/978-3-031-19778-9_20.

[157] Horia Mania, Aurelia Guy, and Benjamin Recht. "Simple Random Search of Static Linear Policies is Competitive for Reinforcement Learning". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montréal, Canada: Curran Associates Inc., 2018, pp. 1805–1814.

[158] Jon McCormack and Camilo Cruz Gambardella. "Quality-Diversity for Aesthetic Evolution". In: *Artificial Intelligence in Music, Sound, Art and Design*. Ed. by Tiago Martins, Nereida Rodríguez-Fernández, and Sérgio M. Rebelo. Cham: Springer International Publishing, 2022, pp. 369–384. ISBN: 978-3-031-03789-4.

[159] Timothy Merino, M Charity, and Julian Togelius. "Interactive Latent Variable Evolution for the Generation of Minecraft Structures". In: *Proceedings of the 18th International Conference on the Foundations of Digital Games*. FDG '23. Lisbon, Portugal: Association for Computing Machinery, 2023. ISBN: 9781450398558. DOI: 10.1145/3582437.3587208.

[160] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous Methods for Deep Reinforcement Learning". In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. Ed. by Maria-Florina Balcan and Kilian Q. Weinberger. Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, 2016, pp. 1928–1937. URL: http://proceedings.mlr.press/v48/mniha16.html.

[161] Thomas M. Moerland, Joost Broekens, and Catholijn M. Jonker. "Model-based Reinforcement Learning: A Survey". In: *CoRR* abs/2006.16712 (2020). URL: https://arxiv.org/abs/2006.16712.

[162] Ouaguenouni Mohamed. *Quality-Diversity Algorithms: MAP-Polar*. https://towardsdatascience.com/quality-diversity-algorithms-a-new-approach-based-on-map-elites-applied-to-robot-navigation-f51380deec5d. Retrieved 2023-01-31. Mar. 2021.

[163] Matej Moravcik, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael H. Bowling. "DeepStack: Expert-Level Artificial Intelligence in No-Limit Poker". In: *Science* (2017).

[164] Aurélien Morel, Yakumo Kunimoto, Alex Coninx, and Stéphane Doncieux. "Automatic Acquisition of a Repertoire of Diverse Grasping Trajectories through Behavior Shaping and Novelty Search". In: *2022 International Conference on Robotics and Automation (ICRA)*. 2022, pp. 755–761. DOI: 10.1109/ICRA46639.2022.9811837.

[165] Douglas Morrison, Peter Corke, and Jürgen Leitner. "EGAD! An Evolved Grasping Analysis Dataset for Diversity and Reproducibility in Robotic Manipulation". In: *IEEE Robotics and Automation Letters* 5.3 (2020), pp. 4368–4375. DOI: 10.1109/LRA.2020.2992195.

[166] Kyriacos Mosphilis and Vassilis Vassiliades. "Optimizing Camera Placement for Chicken Farm Monitoring". In: *Applications of Evolutionary Computation*. Ed. by Pablo García-Sánchez, Emma Hart, and Sarah L. Thomson. Cham: Springer Nature Switzerland, 2025, pp. 354–369. ISBN: 978-3-031-90062-4.

[167] J.-B. Mouret and S. Doncieux. "SFERESv2: Evolvin' in the Multi-Core World". In: *Proc. of Congress on Evolutionary Computation (CEC)*. 2010, pp. 4079–4086.

[168] Jean-Baptiste Mouret et al. *Python3 Map-Elites*. https://github.com/resibots/pymap_elites. 2019.

[169] Jean-Baptiste Mouret and Jeff Clune. "Illuminating search spaces by mapping elites". In: *arXiv preprint arXiv:1504.04909* (2015).

[170] Jean-Baptiste Mouret and Glenn Maguire. "Quality diversity for multi-task optimization". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. ACM, June 2020. DOI: 10.1145/3377930.3390203.

[171] Nils Müller and Tobias Glasmachers. "Challenges in High-Dimensional Reinforcement Learning with Evolution Strategies". In: *Parallel Problem Solving from Nature - PPSN XV - 15th International Conference, Coimbra, Portugal, September 8-12, 2018, Proceedings, Part II*. Ed. by Anne Auger, Carlos M. Fonseca, Nuno Lourenço, Penousal Machado, Luis Paquete, and L. Darrell Whitley. Vol. 11102. Lecture Notes in Computer Science. Springer, 2018, pp. 411–423. DOI: 10.1007/978-3-319-99259-4_33.

[172] Galen E. Mullins, Paul G. Stankiewicz, R. Chad Hawthorne, and Satyandra K. Gupta. "Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles". In: *Journal of Systems and Software* (2018). DOI: 10.1016/j.jss.2017.10.031.

[173] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping". In: *Proceedings of the Sixteenth International Conference on Machine Learning*. ICML '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 278–287. ISBN: 1558606122.

[174] Olle Nilsson. *QDgym*. https://github.com/ollenilsson19/QDgym. 2021.

[175] Olle Nilsson and Antoine Cully. "Policy gradient assisted MAP-Elites". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '21. Lille, France: Association for Computing Machinery, 2021, pp. 866–875. ISBN: 9781450383509. DOI: 10.1145/3449639.3459304.

[176] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. "Solving Rubik's Cube with a Robot Hand". In: *arXiv preprint* (2019).

[177] Misha Paauw and Daan Van den Berg. "Paintings, polygons and plant propagation". In: *International Conference on Computational Intelligence in Music, Sound, Art and Design (Part of EvoStar)*. Springer. 2019, pp. 84–97. URL: https://link.springer.com/chapter/10.1007/978-3-030-16667-0_6.

[178] Paolo Pagliuca, Nicola Milano, and Stefano Nolfi. "Efficacy of Modern Neuro-Evolutionary Strategies for Continuous Control Optimization". In: *Frontiers in Robotics and AI* 7 (2020), p. 98. ISSN: 2296-9144. DOI: 10.3389/frobt.2020.00098.

[179] Giuseppe Paolo, Alexandre Coninx, Stephane Doncieux, and Alban Laflaquière. "Sparse Reward Exploration via Novelty Search and Emitters". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '21. Lille, France: Association for Computing Machinery, 2021, pp. 154–162. ISBN: 9781450383509. DOI: 10.1145/3449639.3459314.

[180] Giuseppe Paolo, Alban Laflaquière, Alexandre Coninx, and Stephane Doncieux. "Unsupervised Learning and Exploration of Reachable Outcome Space". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 2379–2385. DOI: 10.1109/ICRA40945.2020.9196819.

[181] Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob N. Foerster, Edward Grefenstette, and Tim Rocktäschel. "Evolving Curricula with Regret-Based Environment Design". In: *CoRR* abs/2203.01302 (2022). DOI: 10.48550/arXiv.2203.01302.

[182] Jack Parker-Holder, Aldo Pacchiano, Krzysztof M Choromanski, and Stephen J Roberts. "Effective Diversity in Population Based Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 18050–18062. URL: https://proceedings.neurips.cc/paper/2020/file/d1dc3a8270a6f9394f88847d7f0050cf-Paper.pdf.

[183] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

[184] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 3803–3810. DOI: 10.1109/ICRA.2018.8460528.

[185] Malsha V. Perera and Vishal M. Patel. "Analyzing Bias in Diffusion-based Face Generation Models". In: *2023 IEEE International Joint Conference on Biometrics (IJCB)*. 2023, pp. 1–10. DOI: 10.1109/IJCB57857.2023.10449200.

[186] Thomas Pierrot, Valentin Macé, Felix Chalumeau, Arthur Flajolet, Geoffrey Cideron, Karim Beguir, Antoine Cully, Olivier Sigaud, and Nicolas Perrin-Gilbert. "Diversity Policy Gradient for Sample Efficient Quality-Diversity Optimization". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '22. Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 1075–1083. ISBN: 9781450392372. DOI: 10.1145/3512290.3528845.

[187] Antoine Plumerault, Hervé Le Borgne, and Céline Hudelot. “Controlling generative models with continuous factors of variations”. In: *International Conference on Machine Learning (ICLR)*. 2020. URL: https://openreview.net/forum?id=H1laeJrKDB.

[188] Pourchot and Sigaud. “CEM-RL: Combining evolutionary and gradient-based methods for policy search”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=BkeU5j0ctQ.

[189] Justin K. Pugh, L. B. Soros, Paul A. Szerlip, and Kenneth O. Stanley. “Confronting the Challenge of Quality Diversity”. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. GECCO ’15. Madrid, Spain: Association for Computing Machinery, 2015, pp. 967–974. ISBN: 9781450334723. DOI: 10.1145/2739480.2754664.

[190] Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. “Quality Diversity: A New Frontier for Evolutionary Computation”. In: *Frontiers in Robotics and AI* 3 (2016), p. 40. ISSN: 2296-9144. DOI: 10.3389/frobt.2016.00040.

[191] Python Code Quality Authority. *pylint*. http://pylint.pycqa.org/. Computer software.

[192] Python Software Foundation. *Python Package Index*. https://pypi.org.

[193] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. “Learning Transferable Visual Models From Natural Language Supervision”. In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 18–24 Jul 2021, pp. 8748–8763. URL: https://proceedings.mlr.press/v139/radford21a.html.

[194] Md Masudur Rahman and Yexiang Xue. *Robust Policy Optimization in Deep Reinforcement Learning*. 2022. arXiv: 2212.07536 [cs.LG]. URL: https://arxiv.org/abs/2212.07536.

[195] Nemanja Rakicevic, Antoine Cully, and Petar Kormushev. “Policy manifold search: exploring the manifold hypothesis for diversity-based neuroevolution”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO ’21. Lille, France: Association for Computing Machinery, 2021, pp. 901–909. ISBN: 9781450383509. DOI: 10.1145/3449639.3459320.

[196] Sebastian Risi and Julian Togelius. “Increasing generality in machine learning through procedural content generation”. In: *Nature Machine Intelligence* (2020).

[197] E. Rocklage, H. Kraft, A. Karatas, and J. Seewig. “Automated scenario generation for regression testing of autonomous vehicles”. In: *Proceedings of the 20th IEEE International Conference on Intelligent Transportation Systems (ITSC)*. 2017. DOI: 10.1109/ITSC.2017.8317919.

[198] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. “High-Resolution Image Synthesis With Latent Diffusion Models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 10684–10695.

[199] Raymond Ros and Nikolaus Hansen. “A Simple Modification in CMA-ES Achieving Linear Time and Space Complexity”. In: *Parallel Problem Solving from Nature, PPSN X*. Ed. by Günter Rudolph, Thomas Jansen, Nicola Beume, Simon Lucas, and Carlo Poloni. Springer Berlin Heidelberg, 2008, pp. 296–305. ISBN: 978-3-540-87700-4.

[200] Aditi Roy, Nasir Memon, Julian Togelius, and Arun Ross. “Evolutionary methods for generating synthetic masterprint templates: Dictionary attack in fingerprint recognition”. In: *2018 International Conference on Biometrics (ICB)*. IEEE. 2018, pp. 39–46.

[201] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. "Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning". In: *Conference on Robot Learning, 8-11 November 2021, London, UK*. Ed. by Aleksandra Faust, David Hsu, and Gerhard Neumann. Vol. 164. Proceedings of Machine Learning Research. PMLR, 2021, pp. 91–100. URL: https://proceedings.mlr.press/v164/rudin22a.html.

[202] Fereshteh Sadeghi and Sergey Levine. "CAD2RL: Real Single-Image Flight Without a Single Real Image". In: *Proceedings of Robotics: Science and Systems XIII*. 2017. DOI: 10.15607/RSS.2017.XIII.034.

[203] Dorsa Sadigh, S Shankar Sastry, and Sanjit A Seshia. "Verifying robustness of human-aware autonomous cars". In: *IFAC-PapersOnLine* (2019).

[204] Achkan Salehi, Alexandre Coninx, and Stephane Doncieux. "BR-NS: an archive-less approach to novelty search". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '21. Lille, France: Association for Computing Machinery, 2021, pp. 172–179. ISBN: 9781450383509. DOI: 10.1145/3449639.3459303.

[205] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. *Evolution Strategies as a Scalable Alternative to Reinforcement Learning*. 2017. arXiv: 1703.03864 [stat.ML].

[206] Mikayel Samvelyan, Sharath Chandra Raparthy, Andrei Lupu, Eric Hambro, Aram H. Markosyan, Manish Bhatt, Yuning Mao, Minqi Jiang, Jack Parker-Holder, Jakob Foerster, Tim Rocktäschel, and Roberta Raileanu. "Rainbow teaming: open-ended generation of diverse adversarial prompts". In: *Proceedings of the 38th International Conference on Neural Information Processing Systems*. NIPS '24. Vancouver, BC, Canada: Curran Associates Inc., 2025. ISBN: 9798331314385.

[207] Anurag Sarkar and Seth Cooper. "Generating and Blending Game Levels via Quality-Diversity in the Latent Space of a Variational Autoencoder". In: *Proceedings of the 16th International Conference on the Foundations of Digital Games*. FDG '21. Montreal, QC, Canada: Association for Computing Machinery, 2021. ISBN: 9781450384223. DOI: 10.1145/3472538.3472545.

[208] Anurag Sarkar and Seth Cooper. "Generating and blending game levels via quality-diversity in the latent space of a variational autoencoder". In: *Proceedings of the 16th International Conference on the Foundations of Digital Games*. 2021, pp. 1–11.

[209] Anurag Sarkar, Zhihan Yang, and Seth Cooper. "Conditional Level Generation and Game Blending". In: *CoRR* abs/2010.07735 (2020). URL: https://arxiv.org/abs/2010.07735.

[210] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. "Universal Value Function Approximators". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1312–1320. URL: https://proceedings.mlr.press/v37/schaul15.html.

[211] Lennart Schneider, Florian Pfisterer, Janek Thomas, and Bernd Bischl. "A Collection of Quality Diversity Optimization Problems Derived from Hyperparameter Optimization of Machine Learning Models". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '22. Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 2136–2142. ISBN: 9781450392686. DOI: 10.1145/3520304.3534003.

[212] Jacob Schrum and Cody Crosby. "A Quality Diversity Approach to Evolving Model Rockets". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY, USA: Association for Computing Machinery, 2025, pp. 1471–1479. ISBN: 9798400714658. URL: https://doi.org/10.1145/3712256.3726338.

[213] Jacob Schrum, Vanessa Volz, and Sebastian Risi. “Cppn2gan: Combining compositional pattern producing networks and gans for large-scale pattern generation”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 2020, pp. 139–147.

[214] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. “Trust Region Policy Optimization”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1889–1897. URL: https://proceedings.mlr.press/v37/schulman15.html.

[215] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: http://arxiv.org/abs/1506.02438.

[216] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. “Proximal Policy Optimization Algorithms”. In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[217] Noor Shaker, Julian Togelius, and Mark J. Nelson. *Procedural Content Generation in Games*. Springer, 2016. DOI: 10.1007/978-3-319-42716-4.

[218] Shawn Shan, Jenna Cryan, Emily Wenger, Haitao Zheng, Rana Hanocka, and Ben Y. Zhao. “Glaze: protecting artists from style mimicry by text-to-image models”. In: *Proceedings of the 32nd USENIX Conference on Security Symposium*. SEC ’23. Anaheim, CA, USA: USENIX Association, 2023. ISBN: 978-1-939133-37-3.

[219] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. “Interpreting the Latent Space of GANs for Semantic Face Editing”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Seattle, WA, USA: IEEE, 2020, pp. 9240–9249. ISBN: 978-1-7281-7169-2. DOI: 10.1109/CVPR42600.2020.00926.

[220] Yujun Shen, Ceyuan Yang, Xiaoou Tang, and Bolei Zhou. “InterFaceGAN: Interpreting the Disentangled Face Representation Learned by GANs”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.4 (2022), pp. 2004–2018. DOI: 10.1109/TPAMI.2020.3034267.

[221] Yujun Shen and Bolei Zhou. “Closed-Form Factorization of Latent Semantics in GANs”. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Nashville, TN, USA: IEEE, 2021. ISBN: 9781665445108. DOI: 10.1109/CVPR46437.2021.00158.

[222] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* (2016). DOI: 10.1038/nature16961.

[223] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* (2018).

[224] Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. “Defining and characterizing reward hacking”. In: *Proceedings of the 36th International Conference on Neural Information Processing Systems*. NIPS ’22. New Orleans, LA, USA: Curran Associates Inc., 2022. ISBN: 9781713871088.

[225] Ivan Skorokhodov, Grigorii Sotnikov, and Mohamed Elhoseiny. “Aligning Latent and Image Spaces to Connect the Unconnectable”. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 14124–14133. DOI: 10.1109/ICCV48922.2021.01388.

[226] Sam Snodgrass and Santiago Ontañón. “Experiments in map generation using Markov chains”. In: *Proceedings of the 9th International Conference on the Foundations of Digital Games, FDG*. 2014. URL: http://www.fdg2014.org/papers/fdg2014%5C_paper%5C_29.pdf.

[227] Siddharth Srikanth, Varun Bhatt, Boshen Zhang, Werner Hager, Charles Michael Lewis, Katia P. Sycara, Aaquib Tabrez, and Stefanos Nikolaidis. *Algorithmic Prompt Generation for Diverse Human-like Teaming and Communication with Large Language Models*. 2025. arXiv: 2504.03991 [cs.CL]. URL: https://arxiv.org/abs/2504.03991.

[228] Kenneth O Stanley, Joel Lehman, and Lisa Soros. *Open-endedness: The last grand challenge you've never heard of*. 2017. URL: https://www.oreilly.com/radar/open-endedness-the-last-grand-challenge-youve-never-heard-of/.

[229] Kenneth O Stanley and Risto Miikkulainen. “Evolving neural networks through augmenting topologies”. In: *Evolutionary computation* 10.2 (2002), pp. 99–127.

[230] Kirby Steckel and Jacob Schrum. “Illuminating the space of beatable lode runner levels produced by various generative adversarial networks”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2021, pp. 111–112.

[231] Nathan Sturtevant, Nicolas Decroocq, Aaron Tripodi, and Matthew Guzdial. “The unexpected consequence of incremental design changes”. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. 2020.

[232] Adam Summerville and Michael Mateas. “Super Mario as a String: Platformer Level Generation Via LSTMs”. In: *Proceedings of the First Joint International Conference of Digital Games Research Association and Foundation of Digital Games, DiGRA/FDG*. 2016. URL: http://www.digra.org/digital-library/publications/super-mario-as-a-string-platformer-level-generation-via-lstms/.

[233] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. “Procedural Content Generation via Machine Learning (PCGML)”. In: *IEEE Transactions on Games* (2018). DOI: 10.1109/TG.2018.2846639.

[234] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[235] Takumi Tanabe, Kazuto Fukuchi, Jun Sakuma, and Youhei Akimoto. “Level generation for angry birds with sequential VAE and latent variable evolution”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '21. Lille, France: Association for Computing Machinery, 2021, pp. 1052–1060. ISBN: 9781450383509. DOI: 10.1145/3449639.3459290.

[236] Yunhao Tang. “Guiding Evolutionary Strategies with Off-Policy Actor-Critic”. In: *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '21. Virtual Event, United Kingdom: International Foundation for Autonomous Agents and Multiagent Systems, 2021, pp. 1317–1325. ISBN: 9781450383073.

[237] Yingtao Tian and David Ha. “Modern Evolution Strategies for Creativity: Fitting Concrete Images and Abstract Concepts”. In: *Artificial Intelligence in Music, Sound, Art and Design: 11th International Conference, EvoMUSART 2022, Held as Part of EvoStar 2022, Madrid, Spain, April 20–22, 2022, Proceedings*. Madrid, Spain: Springer-Verlag, 2022, pp. 275–291. ISBN: 978-3-031-03788-7. DOI: 10.1007/978-3-031-03789-4_18.

[238] Bryon Tjanaka, Matthew C Fontaine, David H Lee, Yulun Zhang, Nivedit Reddy Balam, Nathaniel Dennler, Sujay S Garlanka, Nikitas Dimitri Klapsis, and Stefanos Nikolaidis. "Pyribs: A Bare-Bones Python Library for Quality Diversity Optimization". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 220–229. ISBN: 9798400701191. DOI: 10.1145/3583131.3590374.

[239] Bryon Tjanaka, Matthew C Fontaine, Julian Togelius, and Stefanos Nikolaidis. "Approximating gradients for differentiable quality diversity in reinforcement learning". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2022, pp. 1102–1111.

[240] Bryon Tjanaka, Matthew C. Fontaine, David H. Lee, Aniruddha Kalkar, and Stefanos Nikolaidis. "Training Diverse High-Dimensional Controllers by Scaling Covariance Matrix Adaptation MAP-Annealing". In: *IEEE Robotics and Automation Letters* 8.10 (2023), pp. 6771–6778. DOI: 10.1109/LRA.2023.3313012.

[241] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 23–30. DOI: 10.1109/IROS.2017.8202133.

[242] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[243] Julian Togelius, Sergey Karakovskiy, and Robin Baumgarten. "The 2009 Mario AI Competition". In: *Proceedings of the IEEE Congress on Evolutionary Computation, CEC*. 2010. DOI: 10.1109/CEC.2010.5586133.

[244] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. "Search-based procedural content generation: A taxonomy and survey". In: *IEEE Transactions on Computational Intelligence and AI in Games* (2011).

[245] Ruben Rodriguez Torrado, Ahmed Khalifa, Michael Cerny Green, Niels Justesen, Sebastian Risi, and Julian Togelius. "Bootstrapping Conditional GANs for Video Game Level Generation". In: *Proceedings of the IEEE Conference on Games*. 2020.

[246] Hung-Yu Tseng, Lu Jiang, Ce Liu, Ming-Hsuan Yang, and Weilong Yang. "Regularizing Generative Adversarial Networks under Limited Data". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 7917–7927. DOI: 10.1109/CVPR46437.2021.00783.

[247] Konstantinos Varelas, Anne Auger, Dimo Brockhoff, Nikolaus Hansen, Ouassim Ait ElHara, Yann Semet, Rami Kassab, and Frédéric Barbaresco. "A Comparative Study of Large-Scale Variants of CMA-ES". In: *Parallel Problem Solving from Nature, PPSN XV*. Ed. by Anne Auger, Carlos M. Fonseca, Nuno Lourenço, Penousal Machado, Luís Paquete, and Darrell Whitley. Cham: Springer International Publishing, 2018, pp. 3–15. ISBN: 978-3-319-99253-2.

[248] Vassilis Vassiliades, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. "Using Centroidal Voronoi Tessellations to Scale Up the Multidimensional Archive of Phenotypic Elites Algorithm". In: *IEEE Transactions on Evolutionary Computation* 22.4 (2018), pp. 623–630. DOI: 10.1109/TEVC.2017.2735550.

[249] Vassilis Vassiliades and Jean-Baptiste Mouret. "Discovering the elite hypervolume by leveraging interspecies correlation". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2018, pp. 149–156.

[250] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964.

[251] Jonas Verhellen and Jeriek Van den Abeele. "Illuminating elite patches of chemical space". In: *Chem. Sci.* 11 (42 2020), pp. 11485–11491. DOI: 10.1039/D0SC03544K.

[252] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. "Grandmaster Level in StarCraft II Using Multi-Agent Reinforcement Learning". In: *Nature* (2019).

[253] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M. Lucas, Adam Smith, and Sebastian Risi. "Evolving mario levels in the latent space of a deep convolutional generative adversarial network". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '18. Kyoto, Japan: Association for Computing Machinery, 2018, pp. 221–228. ISBN: 9781450356183. DOI: 10.1145/3205455.3205517.

[254] Andrey Voynov and Artem Babenko. "Unsupervised discovery of interpretable directions in the GAN latent space". In: *Proceedings of the 37th International Conference on Machine Learning*. ICML'20. JMLR.org, 2020.

[255] Ren-Jian Wang, Ke Xue, Zeyu Qin, Ziniu Li, Sheng Tang, Hao-Tian Li, Shengcai Liu, and Chao Qian. *Quality-Diversity Red-Teaming: Automated Generation of High-Quality and Diverse Attackers for Large Language Models*. 2025. arXiv: 2506.07121 [cs.LG]. URL: https://arxiv.org/abs/2506.07121.

[256] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. "Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions". In: *CoRR* abs/1901.01753 (2019). URL: http://arxiv.org/abs/1901.01753.

[257] Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeffrey Clune, and Kenneth O. Stanley. "Enhanced POET: Open-ended Reinforcement Learning through Unbounded Invention of Learning Challenges and their Solutions". In: *Proceedings of the 37th International Conference on Machine Learning, ICML*. 2020. URL: http://proceedings.mlr.press/v119/wang20l.html.

[258] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. "Natural Evolution Strategies". In: *Journal of Machine Learning Research* 15.27 (2014), pp. 949–980. URL: http://jmlr.org/papers/v15/wierstra14a.html.

[259] Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. "Natural Evolution Strategies". In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. 2008, pp. 3381–3387. DOI: 10.1109/CEC.2008.4631255.

[260] Dietmar Wolz. *fcmaes - A Python-3 derivative-free optimization library*. Available at https://github.com/dietmarwo/fast-cma-es. Python/C++ source code, with description and examples. 2022.

[261] Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. "Outracing champion Gran Turismo drivers with deep reinforcement learning". In: *Nature* (2022).

[262] Chenjun Xiao, Yifan Wu, Chen Ma, Dale Schuurmans, and Martin Müller. "Learning to Combat Compounding-Error in Model-Based Reinforcement Learning". In: *CoRR* abs/1912.11206 (2019). URL: http://arxiv.org/abs/1912.11206.

[263] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms”. In: *arXiv preprint arXiv:1708.07747* (2017).

[264] Ke Xue, Ren-Jian Wang, Pengyi Li, Dong Li, Jianye Hao, and Chao Qian. “Sample-Efficient Quality-Diversity by Cooperative Coevolution”. In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=JDud6zbpFv.

[265] Marvin Zammit, Antonios Liapis, and Georgios Yannakakis. “Seeding Diversity into AI Art”. In: *International Conference on Computational Creativity*. 2022.

[266] Jenny Zhang, Joel Lehman, Kenneth Stanley, and Jeff Clune. “OMNI: Open-endedness via Models of human Notions of Interestingness”. In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=AgM3MzT99c.

[267] Yulun Zhang, Matthew C Fontaine, Amy K Hoover, and Stefanos Nikolaidis. “Deep surrogate assisted map-elites for automated hearthstone deckbuilding”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2022, pp. 158–167.

[268] Shihan Zhao. *CabbageCat’s Blogs*. https://szhaovas.github.io. Retrieved 2023-01-31.

[269] Yilun Zhou, Serena Booth, Nadia Figueroa, and Julie Shah. “RoCUS: Robot Controller Understanding via Sampling”. In: *Proceedings of the Conference on Robot Learning*. 2021. URL: https://proceedings.mlr.press/v164/zhou22a.html.