DIFFERENTIABLE QUALITY DIVERSITY FOR REINFORCEMENT LEARNING BY APPROXIMATING GRADIENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Consider a walking agent that must adapt to damage. To approach this task, we can train a collection of policies and have the agent select a suitable policy when damaged. Training this collection may be viewed as a quality diversity (QD) optimization problem, where we search for solutions (policies) which maximize an objective (walking forward) while spanning a set of measures (measurable characteristics). Recent work shows that differentiable quality diversity (DQD) algorithms greatly accelerate QD optimization when exact gradients are available for the objective and measures. However, such gradients are typically unavailable in RL settings due to non-differentiable environments. To apply DQD in RL settings, we propose to approximate objective and measure gradients with evolution strategies and actor-critic methods. We develop two variants of the DQD algorithm CMA-MEGA, each with different gradient approximations, and evaluate them on four simulated walking tasks. One variant achieves comparable performance (QD score) with the state-of-the-art PGA-MAP-Elites in two tasks. The other variant performs comparably in all tasks but is less efficient than PGA-MAP-Elites in two tasks. These results provide insight into the limitations of CMA-MEGA in domains that require rigorous optimization of the objective and where exact gradients are unavailable.

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.

Consider a half-cheetah agent (Fig. 2) 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, we must design a reward function that results in a single robustly capable agent. However, prior work (Irpan, 2018; Clark and Amodei, 2016) suggests that designing such a reward function is difficult.

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 (Cully et al., 2015; Colas et al., 2020).

Pre-training such a collection of policies may be viewed as a quality diversity (QD) optimization problem (Pugh et al., 2016; Cully et al., 2015; Mouret and Clune, 2015; Colas et al., 2020). 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 (Fontaine and Nikolaidis, 2021) proposes differentiable quality diversity (DQD), which assumes f and $\boldsymbol{m}$ are differentiable functions with exact gradient information.

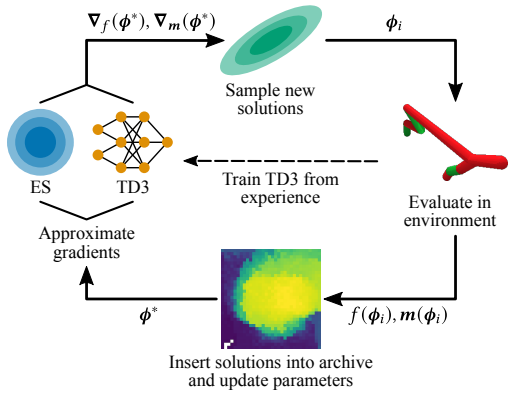


Figure 1: CMA-MEGA (ES) and CMA-MEGA (TD3, ES). Each algorithm first approximates gradients around a mean solution ϕ^*. Then, each algorithm generates new solutions ϕ_i, evaluates them in the environment, and inserts them into an archive. Finally, internal parameters such as ϕ^* are updated based on how each ϕ_i improves the archive. CMA-MEGA (TD3, ES) also trains TD3 from environment experience.

The recently proposed DQD algorithm CMA-MEGA (Fontaine and Nikolaidis, 2021) 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) (Akimoto et al., 2010; Wierstra et al., 2014; Salimans et al., 2017; Mania et al., 2018) and deep RL actor-critic methods (Schulman et al., 2015; 2017; Lillicrap et al., 2016; Fujimoto et al., 2018) optimize a single 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.*

Our work makes three main contributions. **(1)** We formalize the problem of quality diversity for reinforcement learning (QD-RL) and reduce it to an instance of DQD (Sec. 2). **(2)** We develop two QD-RL variants of the DQD algorithm CMA-MEGA (Sec. 4). The first variant, CMA-MEGA (ES), approximates objective and measure gradients with ES. The second variant, CMA-MEGA (TD3, ES), approximates the objective gradient with TD3 (Fujimoto et al., 2018) (an actor-critic method) and the measure gradients with ES. **(3)** We benchmark our variants on four PyBullet locomotion tasks from QDGym (Ellenberger, 2019; Nilsson, 2021) (Sec. 5-6). The first variant, CMA-MEGA (ES), achieves a QD score (Sec. 5.1.3) comparable to the state-of-the-art PGA-MAP-Elites in two tasks. The second variant, CMA-MEGA (TD3, ES), achieves comparable QD score with PGA-MAP-Elites in all tasks but is less efficient than PGA-MAP-Elites in two tasks.

Our results contrast with prior work (Fontaine and Nikolaidis, 2021) where CMA-MEGA vastly outperforms OG-MAP-Elites, a DQD algorithm inspired by PGA-MAP-Elites, 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.

2 Problem Statement

2.1 Quality Diversity (QD)

We adopt the definition of QD from prior work (Fontaine and Nikolaidis, 2021). For a solution vector $\boldsymbol{\phi}$, QD considers an objective function $f(\boldsymbol{\phi})$ and k measures[1] $m_i(\boldsymbol{\phi}) \in \mathbb{R}$ (for $i \in 1..k$) or, as a joint measure, $\boldsymbol{m}(\boldsymbol{\phi}) \in \mathbb{R}^k$. These measures form a k-dimensional measure space $\mathcal{X}$. For

[1]Prior work has also referred to measures as "behavior characteristics" or "behavior descriptors."

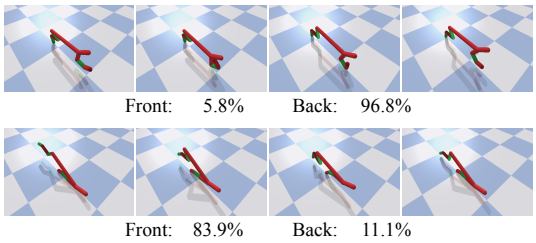


Figure 2: A half-cheetah agent executing two walking policies. In the top row, the agent walks on its back foot while tapping the ground with its front foot. In the bottom row, the agent walks on its front foot while jerking its back foot. Values below each row show the percentage of time each foot contacts 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.

every $\boldsymbol{x} \in \mathcal{X}$, the QD objective is to find solution $\boldsymbol{\phi}$ such that $\boldsymbol{m}(\boldsymbol{\phi}) = \boldsymbol{x}$ and $f(\boldsymbol{\phi})$ is maximized. Since $\mathcal{X}$ is continuous, it would require infinite memory to solve the QD problem, so algorithms in the MAP-Elites family (Mouret and Clune, 2015; Cully et al., 2015) discretize $\mathcal{X}$ by forming a tesselation $\mathcal{Y}$ consisting of M cells. Thus, we relax the QD 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:

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

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 $\boldsymbol{\nabla}_f(\boldsymbol{\phi})$ and the measure gradients as $\boldsymbol{\nabla}_{\boldsymbol{m}}(\boldsymbol{\phi})$ or $\boldsymbol{\nabla}_{m_i}(\boldsymbol{\phi})$.

2.2 Quality Diversity for Reinforcement Learning (QD-RL)

We define QD-RL as an instance of the QD problem in which the objective $f(\boldsymbol{\phi})$ is the *expected discounted return* of an RL policy, and the measures $\boldsymbol{m}(\boldsymbol{\phi})$ are functions of the policy. Formally, drawing on the Markov Decision Process (MDP) formulation (Sutton and Barto, 2018), 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 $\pi_{\boldsymbol{\phi}}(a|s)$ with parameters $\boldsymbol{\phi}$. 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

$$f(\boldsymbol{\phi}) = \mathbb{E}_{\xi \sim p_{\boldsymbol{\phi}}} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right] \tag{2}$$

where the discount factor $\gamma \in (0, 1)$ trades off between short- and long-term rewards. Finally, each policy is characterized by a k-dimensional measure function $\boldsymbol{m}(\boldsymbol{\phi})$.

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 BACKGROUND

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. 2, with the goal of applying these algorithms' gradient approximations to DQD in Sec. 4.

3.1.1 EVOLUTION STRATEGIES (ES)

ES (Beyer and Schwefel, 2002) 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) (Wierstra et al., 2014; 2008) 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 (Amari, 1998). 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 two NES-inspired approaches which have demonstrated success in RL domains. First, prior work (Salimans et al., 2017) introduces an algorithm which, drawing from NES, samples λ_{es} solutions from an isotropic Gaussian but only computes a gradient step for the mean $\boldsymbol{\phi}$. We refer to this algorithm as OpenAI-ES. 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

$$\boldsymbol{\nabla}_f(\boldsymbol{\phi}) \approx \frac{1}{\lambda_{es}\sigma} \sum_{i=1}^{\lambda_{es}} f(\boldsymbol{\phi} + \sigma\boldsymbol{\epsilon}_i)\boldsymbol{\epsilon}_i \tag{3}$$

OpenAI-ES then passes this estimate to an Adam optimizer (Kingma and Ba, 2015) 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 (Ha, 2017; Wierstra et al., 2014).

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) (Fujimoto et al., 2018), 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.

3.2 QUALITY DIVERSITY ALGORITHMS

3.2.1 MAP-ELITES EXTENSIONS FOR QD-RL

One of the simplest QD algorithms is MAP-Elites (Mouret and Clune, 2015; Cully et al., 2015). 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 $\mathcal{N}(\boldsymbol{\phi}_{\mathbf{0}}, \sigma\boldsymbol{I})$ 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, MAP-Elites has not proven effective for QD-RL. Hence, several extensions merge MAP-Elites with actor-critic methods and ES. For

instance, Policy Gradient Assisted MAP-Elites (PGA-MAP-Elites) (Nilsson and Cully, 2021) combines MAP-Elites with TD3. Each iteration, PGA-MAP-Elites 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 (Vassiliades and Mouret, 2018) which selects two solutions $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$ from the archive and creates a new one according to $\boldsymbol{\phi}' = \boldsymbol{\phi}_1 + \sigma_1 \mathcal{N}(\mathbf{0}, \boldsymbol{I}) + \sigma_2(\boldsymbol{\phi}_2 - \boldsymbol{\phi}_1)\mathcal{N}(0, 1)$. PGA-MAP-Elites achieves state-of-the-art performance on locomotion tasks in the QDGym benchmark (Nilsson, 2021).

Another MAP-Elites extension is ME-ES (Colas et al., 2020), 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 at a mean $\boldsymbol{\phi}$ and optimizes the objective for k iterations, inserting the current $\boldsymbol{\phi}$ 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 (Hansen, 2016) instance to decide which gradients to follow to maximize QD score. We include MAP-Elites, PGA-MAP-Elites, and ME-ES as baselines in our experiments.

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

We directly extend CMA-MEGA (Fontaine and Nikolaidis, 2021) to address QD-RL. CMA-MEGA is a DQD algorithm based on the QD algorithm CMA-ME (Fontaine et al., 2020). The intuition behind CMA-MEGA is that for solution $\boldsymbol{\phi}$, we can simultaneously increase/decrease both the objective and measures by following the objective and measure gradients of $\boldsymbol{\phi}$. In doing so, we traverse *objective-measure space* and generate solutions which maximize improvement of the archive.

Each iteration, CMA-MEGA first calculates objective and measure gradients for a mean solution $\boldsymbol{\phi}^*$. Next, it generates λ new solutions by sampling gradient coefficients $\boldsymbol{c} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and computing $\boldsymbol{\phi}' \leftarrow \boldsymbol{\phi}^* + \boldsymbol{c}_0 \boldsymbol{\nabla}_f(\boldsymbol{\phi}^*) + \sum_{j=1}^{k} \boldsymbol{c}_j \boldsymbol{\nabla}_{m_j}(\boldsymbol{\phi}^*)$. 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. Finally, CMA-MEGA passes the ranking to CMA-ES (Hansen, 2016), which adapts $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ such that future gradient coefficients $\boldsymbol{c}$ are more likely to generate archive improvement. By leveraging gradient information, CMA-MEGA solves QD benchmarks with orders of magnitude fewer solution evaluations than previous QD algorithms.

4 Approximating Gradients for CMA-MEGA

Since CMA-MEGA requires exact objective and measure gradients, we cannot directly apply it to QD-RL. To address this limitation, we replace exact gradients with gradient approximations (Sec. 4.1) and develop two CMA-MEGA variants (Sec. 4.2).

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.

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, i.e. the OpenAI-ES gradient estimate in Eq. 3. Since OpenAI-ES performs well in RL domains (Salimans et al., 2017; Pagliuca et al., 2020; Lehman et al.,

2018), 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. 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 (Fujimoto et al., 2018) shows that TD3 outperforms on-policy methods (Schulman et al., 2015; 2017). While the off-policy Soft Actor-Critic (Haarnoja et al., 2018) algorithm can outperform TD3, it optimizes a maximum-entropy objective designed to encourage exploration. In our work, this exploration is unnecessary because QD algorithms already search for diverse solutions.

4.1.2 Approximating measure gradients with ES

Since we treat measures as black boxes (Sec. 2.2), we can only estimate their gradient with black box methods. Thus, similar to the objective, we approximate each individual measure's gradient $\boldsymbol{\nabla}_{m_i}$ with the OpenAI-ES gradient estimate, replacing f with m_i in Eq. 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. 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.

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 the measure gradients with OpenAI-ES. Refer to Appendix B for a detailed explanation of these algorithms.

5 Experiments

We compare our two proposed CMA-MEGA variants (CMA-MEGA (ES), CMA-MEGA (TD3, ES)) with three baselines (PGA-MAP-Elites, ME-ES, MAP-Elites) in four locomotion tasks. We implement MAP-Elites as described in Sec. 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 (Colas et al., 2020).

5.1 Evaluation Domains

5.1.1 QDGym

We evaluate our algorithms in four locomotion environments from QDGym (Nilsson, 2021), a library built on PyBullet Gym (Coumans and Bai, 2020; Ellenberger, 2019) and OpenAI Gym (Brockman et al., 2016). Table 6, Appendix D 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 also reward shaping (Ng et al., 1999) 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.

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 (Glorot and Bengio, 2010). For the archive, we tesselate each environment's measure space into a grid of evenly-sized cells (see Table 6 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 Appendix C for further hyperparameters.

5.1.3 METRICS

Our primary metric is *QD score* (Pugh et al., 2016), 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 set the objective f to be the *expected undiscounted return*, i.e. we set $\gamma = 1$ in Eq. 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.

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-MAP-Elites, 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. We test three hypotheses: **[H1]** In each environment, CMA-MEGA (ES) will outperform (with respect to QD score) all baselines (PGA-MAP-Elites, ME-ES, MAP-Elites). **[H2]** In each environment, CMA-MEGA (TD3, ES) will outperform all baselines. **[H3]** In each environment, CMA-MEGA (TD3, ES) will outperform CMA-MEGA (ES).

H1 and H2 are based on prior work (Fontaine and Nikolaidis, 2021) 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 (Pagliuca et al., 2020) 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.

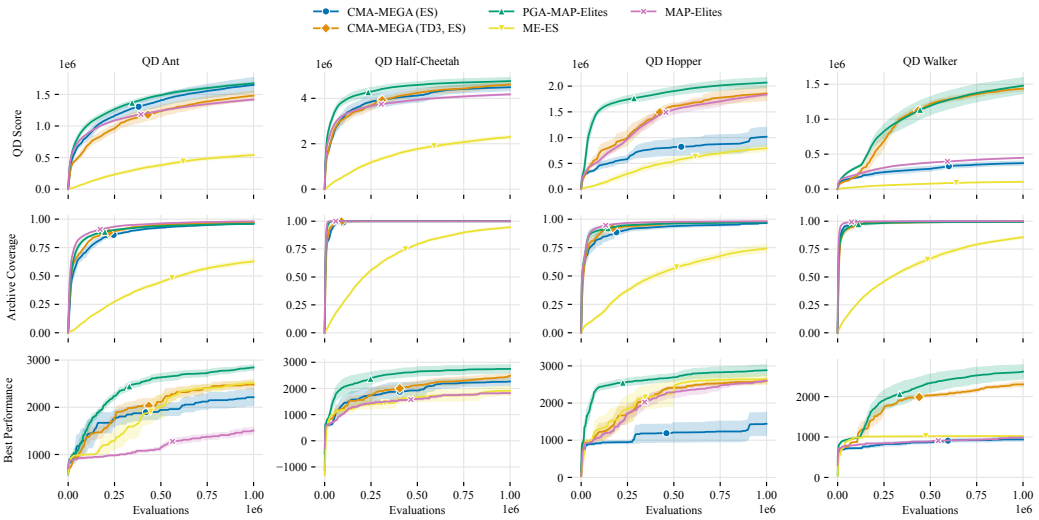


Figure 3: Plots of QD score, archive coverage, and best performance for the 5 algorithms in our experiments in all 4 environments from QDGym. The x-axis in all plots is the number of solutions evaluated. Solid lines show the mean over 5 trials, and shaded regions show the standard error of the mean.

5.3 Implementation

We implement all QD algorithms with the pyribs library (Tjanaka et al., 2021) 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-MAP-Elites). Depending on the algorithm and environment, each experiment lasts 4-20 hours; refer to Table 12, Appendix E for mean runtimes.

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 plots the metrics from our experiments, and Appendix E lists final values of all metrics. Appendix H shows example heatmaps and histograms of each archive, and the supplemental material contains videos of generated agents. Refer to Appendix G for a discussion of our results.

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-MAP-Elites in all tasks but was less efficient in two of the tasks. CMA-MEGA (ES) performed comparably in two tasks. Since we decouple DQD from gradient approximations, our work opens avenues for future research which expands the applicability of DQD in RL settings, by independently developing either more accurate gradient approximations or more powerful DQD algorithms.

References

Youhei Akimoto, Yuichi Nagata, Isao Ono, and Shigenobu Kobayashi. 2010. Bidirectional Relation between CMA Evolution Strategies and Natural Evolution Strategies. In *Parallel Problem Solving*

from Nature, PPSN XI, Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 154–163.

Shun-ichi Amari. 1998. Natural Gradient Works Efficiently in Learning. *Neural Computation* 10, 2 (02 1998), 251–276. https://doi.org/10.1162/089976698300017746 arXiv:https://direct.mit.edu/neco/article-pdf/10/2/251/813415/089976698300017746.pdf

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

Hans-Georg Beyer and Hans-Paul Schwefel. 2002. Evolution strategies – A comprehensive introduction. *Natural Computing* 1, 1 (01 Mar 2002), 3–52. https://doi.org/10.1023/A:1015059928466

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

Geoffrey Cideron, Thomas Pierrot, Nicolas Perrin, Karim Beguir, and Olivier Sigaud. 2020. QD-RL: Efficient Mixing of Quality and Diversity in Reinforcement Learning. *CoRR* abs/2006.08505 (2020). arXiv:2006.08505 https://arxiv.org/abs/2006.08505

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

Cédric Colas, Vashisht Madhavan, Joost Huizinga, and Jeff Clune. 2020. Scaling MAP-Elites to Deep Neuroevolution. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* (Cancún, Mexico) *(GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 67–75. https://doi.org/10.1145/3377930.3390217

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

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

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

Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. 2015. Robots that can adapt like animals. *Nature* 521 (05 2015), 503–507. https://doi.org/10.1038/nature14422

Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. 2005. A Tutorial on the Cross-Entropy Method. *Annals of Operations Research* 134, 1 (01 Feb 2005), 19–67. https://doi.org/10.1007/s10479-005-5724-z

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

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

Matthew C. Fontaine and Stefanos Nikolaidis. 2021. Differentiable Quality Diversity. *Advances in Neural Information Processing Systems* 34 (2021). https://proceedings.neurips.cc/paper/2021/file/532923f11ac97d3e7cb0130315b067dc-Paper.pdf

Matthew C. Fontaine, Julian Togelius, Stefanos Nikolaidis, and Amy K. Hoover. 2020. Covariance Matrix Adaptation for the Rapid Illumination of Behavior Space. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* (Cancún, Mexico) *(GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 94–102. https://doi.org/10.1145/3377930.3390232

Scott Fujimoto, Herke van Hoof, and David Meger. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, 1587–1596. http://proceedings.mlr.press/v80/fujimoto18a.html

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

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

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

Nikolaus Hansen. 2016. The CMA Evolution Strategy: A Tutorial. *CoRR* abs/1604.00772 (2016). arXiv:1604.00772 http://arxiv.org/abs/1604.00772

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

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

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

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

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

Joel Lehman, Jay Chen, Jeff Clune, and Kenneth O. Stanley. 2018. ES is More than Just a Traditional Finite-Difference Approximator. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Kyoto, Japan) *(GECCO '18)*. Association for Computing Machinery, New York, NY, USA, 450–457. https://doi.org/10.1145/3205455.3205474

Joel Lehman and Kenneth O. Stanley. 2011a. Abandoning Objectives: Evolution Through the Search for Novelty Alone. *Evolutionary Computation* 19, 2 (06 2011), 189–223. https://doi.org/10.1162/EVCO_a_00025 arXiv:https://direct.mit.edu/evco/article-pdf/19/2/189/1494066/evco_a_00025.pdf

Joel Lehman and Kenneth O. Stanley. 2011b. Evolving a Diversity of Virtual Creatures through Novelty Search and Local Competition. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (Dublin, Ireland) *(GECCO '11)*. Association for Computing Machinery, New York, NY, USA, 211–218. https://doi.org/10.1145/2001576.2001606

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

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. 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*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1509.02971

Horia Mania, Aurelia Guy, and Benjamin Recht. 2018. Simple Random Search of Static Linear Policies is Competitive for Reinforcement Learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 1805–1814.

Jean-Baptiste Mouret and Jeff Clune. 2015. Illuminating search spaces by mapping elites. *CoRR* abs/1504.04909 (2015). arXiv:1504.04909 http://arxiv.org/abs/1504.04909

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

Olle Nilsson. 2021. QDgym. https://github.com/ollenilsson19/QDgym.

Olle Nilsson and Antoine Cully. 2021. Policy Gradient Assisted MAP-Elites. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Lille, France) *(GECCO '21)*. Association for Computing Machinery, New York, NY, USA, 866–875. https://doi.org/10.1145/3449639.3459304

Paolo Pagliuca, Nicola Milano, and Stefano Nolfi. 2020. Efficacy of Modern Neuro-Evolutionary Strategies for Continuous Control Optimization. *Frontiers in Robotics and AI* 7 (2020), 98. https://doi.org/10.3389/frobt.2020.00098

Jack Parker-Holder, Aldo Pacchiano, Krzysztof M Choromanski, and Stephen J Roberts. 2020. Effective Diversity in Population Based Reinforcement Learning. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 18050–18062. https://proceedings.neurips.cc/paper/2020/file/d1dc3a8270a6f9394f88847d7f0050cf-Paper.pdf

Pourchot and Sigaud. 2019. CEM-RL: Combining evolutionary and gradient-based methods for policy search. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BkeU5j0ctQ

Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. 2016. Quality Diversity: A New Frontier for Evolutionary Computation. *Frontiers in Robotics and AI* 3 (2016), 40. https://doi.org/10.3389/frobt.2016.00040

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. arXiv:1703.03864 [stat.ML]

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. 2015. Universal Value Function Approximators. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 37)*, Francis Bach and David Blei (Eds.). PMLR, Lille, France, 1312–1320. https://proceedings.mlr.press/v37/schaul15.html

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 37)*, Francis Bach and David Blei (Eds.). PMLR, Lille, France, 1889–1897. https://proceedings.mlr.press/v37/schulman15.html

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). arXiv:1707.06347 http://arxiv.org/abs/1707.06347

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (second ed.). The MIT Press. http://incompleteideas.net/book/the-book-2nd.html

Yunhao Tang. 2021. Guiding Evolutionary Strategies with Off-Policy Actor-Critic. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems* (Virtual Event, United Kingdom) *(AAMAS '21)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 1317–1325.

Bryon Tjanaka, Matthew C. Fontaine, Yulun Zhang, Sam Sommerer, Nathan Dennler, and Stefanos Nikolaidis. 2021. pyribs: A bare-bones Python library for quality diversity optimization. https://github.com/icaros-usc/pyribs.

Vassilis Vassiliades and Jean-Baptiste Mouret. 2018. Discovering the Elite Hypervolume by Leveraging Interspecies Correlation. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Kyoto, Japan) *(GECCO '18)*. Association for Computing Machinery, New York, NY, USA, 149–156. https://doi.org/10.1145/3205455.3205602

Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. 2014. Natural Evolution Strategies. *Journal of Machine Learning Research* 15, 27 (2014), 949–980. http://jmlr.org/papers/v15/wierstra14a.html

Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. 2008. Natural Evolution Strategies. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. 3381–3387. https://doi.org/10.1109/CEC.2008.4631255

A RELATED WORK

A.1 BEYOND MAP-ELITES

Several QD-RL algorithms have been developed outside the MAP-Elites family. NS-ES (Conti et al., 2018) builds on Novelty Search (NS) (Lehman and Stanley, 2011a;b), 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, the QD-RL algorithm (Cideron et al., 2020) (distinct from the QD-RL problem we define) maintains an archive with all past solutions and optimizes agents along a Pareto front of the objective and novelty. Finally, Diversity via Determinants (DvD) (Parker-Holder et al., 2020) 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.

A.2 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 $\pi_{\boldsymbol{\phi}}(a|s, z)$ (Eysenbach et al., 2019; Kumar et al., 2020; Li et al., 2017) or goal-conditioned policy $\pi_{\boldsymbol{\phi}}(a|s, g)$ (Schaul et al., 2015; Andrychowicz et al., 2017), 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, QD-RL algorithms *observe* the measures $\boldsymbol{m}(\boldsymbol{\phi})$ instead of *conditioning* on some $\boldsymbol{m}$. Hence, QD-RL outputs an archive of nonconditioned policies rather than a single conditioned policy.

Another area of work combines evolutionary and actor-critic algorithms to solve single-objective hard-exploration problems (Colas et al., 2018; Khadka and Tumer, 2018; Pourchot and Sigaud, 2019; Tang, 2021; Khadka et al., 2019). In these methods, an evolutionary algorithm such as cross-entropy method (de Boer et al., 2005) 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 diversity via a measure function and stores the solutions in an archive. In contrast, these methods do not need an explicit measure of diversity, as they assume that the policies in their populations are sufficiently "different" that they can drive exploration and discover the optimal single objective.

B CMA-MEGA VARIANTS

B.1 CMA-MEGA PSEUDOCODE

Fig. 1 shows an overview of both algorithms, and Algorithm 1 shows their pseudocode. Since CMA-MEGA (TD3, ES) builds on CMA-MEGA (ES), we include only one algorithm and highlight lines that CMA-MEGA (TD3, ES) additionally executes. The rest of this section reviews the pseudocode in Algorithm 1.

We first set the batch size λ' for CMA-ES (line 2). While CMA-MEGA (ES) and CMA-MEGA (TD3, ES) both have a total batch size of λ solutions on each iteration, one of these solutions is reserved for evaluating $\boldsymbol{\phi}^*$ (line 7). In CMA-MEGA (TD3, ES), an additional solution is reserved for evaluating the greedy actor (line 26). This leaves λ' solutions remaining for the CMA-ES instance.

Next, we initialize various objects (lines 3-4). CMA-MEGA (TD3, ES) also initializes a replay buffer $\mathcal{B}$, critic networks $Q_{\boldsymbol{\theta}_1}$ and $Q_{\boldsymbol{\theta}_2}$, a "greedy" actor $\pi_{\boldsymbol{\phi}_q}$ which is used to train the critics, and target networks $Q_{\boldsymbol{\theta}'_1}$, $Q_{\boldsymbol{\theta}'_2}$, $\pi_{\boldsymbol{\phi}'_q}$ (line 5).

In the main loop (line 6), we first evaluate $\boldsymbol{\phi}^*$ (line 7) and insert it into $\mathcal{A}$ (line 8). Then, we estimate the objective and measure gradients of $\boldsymbol{\phi}^*$ with OpenAI-ES (line 9). This estimate computes all objective and measure gradients after evaluating λ_{es} solutions (Algorithm 4, Appendix B.2). In CMA-MEGA (TD3, ES), we override the objective gradient estimate with the TD3 estimate (line 10). This estimate samples n_{pg} transitions of experience from $\mathcal{B}$ and computes a gradient ascent step with $Q_{\boldsymbol{\theta}_1}$ (Algorithm 3, Appendix B.2). To make this estimate more accurate, we sample many transitions ($n_{pg} = 65,536$ instead of the default of 100 in TD3 (Fujimoto et al., 2018)).

Once the gradients are computed, we normalize them to be unit vectors (line 11) and generate λ' solutions for insertion into $\mathcal{A}$. Specifically, we sample gradient coefficients $\boldsymbol{c}$ (line 13), compute perturbation $\boldsymbol{\nabla}_i$ (line 14), and take a gradient step to obtain $\boldsymbol{\phi}'_i$ (line 15). Finally, we evaluate $\boldsymbol{\phi}'_i$ (line 16) and store the improvement Δ_i (Sec. 3.2.2) from inserting it into $\mathcal{A}$ (line 17).

On line 19, we create an *improvement ranking* for $\boldsymbol{c}_i$ and $\boldsymbol{\nabla}_i$ based on Δ_i. We update the CMA-ES parameters and $\boldsymbol{\phi}^*$ based on the ranking (lines 20-21). If CMA-ES did not generate any solutions that were inserted into $\mathcal{A}$, we reset CMA-ES and $\boldsymbol{\phi}^*$ (lines 22-24).

Finally, we update the TD3 instance in CMA-MEGA (TD3, ES). First, identically to PGA-MAP-Elites, we evaluate and insert the greedy actor $\boldsymbol{\phi}_q$ (lines 26-27). Then, we add experience to $\mathcal{B}$ (line 28), including experience from the OpenAI-ES gradient estimate (line 9), and train the critics (line 29). In practice, this training executes in parallel with the rest of the loop to reduce runtime.

Algorithm 1: CMA-MEGA (ES) and CMA-MEGA (TD3, ES). Highlighted portions are only executed in CMA-MEGA (TD3, ES). Adapted from CMA-MEGA (Fontaine and Nikolaidis, 2021). Refer to Appendix B.2 for SMALL_CAPS methods.

CMA-MEGA $(evaluate, \boldsymbol{\phi}_0, N, \lambda, \sigma_g, \eta, \lambda_{es}, \sigma_e)$**:**

  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 $\mathcal{A}$

  $\lambda' \leftarrow \lambda - 1 \; - 1$

  Initialize empty archive $\mathcal{A}$, mean parameters $\boldsymbol{\phi}^*$ to $\boldsymbol{\phi}_0$

  Initialize CMA-ES with population λ', resulting in $\boldsymbol{\mu} = \mathbf{0}, \boldsymbol{\Sigma} = \sigma_g \boldsymbol{I}$, and internal CMA-ES parameters $\boldsymbol{p}$

  $\mathcal{B}, Q_{\boldsymbol{\theta}_1}, Q_{\boldsymbol{\theta}_2}, \pi_{\boldsymbol{\phi}_q}, Q_{\boldsymbol{\theta}'_1}, Q_{\boldsymbol{\theta}'_2}, \pi_{\boldsymbol{\phi}'_q} \leftarrow$ INITIALIZE_TD3()

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

    $f(\boldsymbol{\phi}^*), \boldsymbol{m}(\boldsymbol{\phi}^*) \leftarrow evaluate(\boldsymbol{\phi}^*)$

    UPDATE_ARCHIVE($\mathcal{A}, \boldsymbol{\phi}^*, f(\boldsymbol{\phi}^*), \boldsymbol{m}(\boldsymbol{\phi}^*)$)

    $\boldsymbol{\nabla}_f(\boldsymbol{\phi}^*), \boldsymbol{\nabla}_{\boldsymbol{m}}(\boldsymbol{\phi}^*) \leftarrow$ ES_GRADIENTS($\boldsymbol{\phi}^*, \lambda_{es}, \sigma_e$)

    $\boldsymbol{\nabla}_f(\boldsymbol{\phi}^*) \leftarrow$ TD3_GRADIENT($\boldsymbol{\phi}^*, Q_{\boldsymbol{\theta}_1}, \mathcal{B}$)

    Normalize $\boldsymbol{\nabla}_f(\boldsymbol{\phi}^*)$ and $\boldsymbol{\nabla}_{\boldsymbol{m}}(\boldsymbol{\phi}^*)$ to be unit vectors

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

      $\boldsymbol{c}_i \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$

      $\boldsymbol{\nabla}_i \leftarrow \boldsymbol{c}_{i,0} \boldsymbol{\nabla}_f(\boldsymbol{\phi}^*) + \sum_{j=1}^{k} \boldsymbol{c}_{i,j} \boldsymbol{\nabla}_{m_j}(\boldsymbol{\phi}^*)$

      $\boldsymbol{\phi}'_i \leftarrow \boldsymbol{\phi}^* + \boldsymbol{\nabla}_i$

      $f(\boldsymbol{\phi}'_i), \boldsymbol{m}'(\boldsymbol{\phi}'_i) \leftarrow evaluate(\boldsymbol{\phi}'_i)$

      $\Delta_i \leftarrow$ UPDATE_ARCHIVE($\mathcal{A}, \boldsymbol{\phi}'_i, f(\boldsymbol{\phi}'_i), \boldsymbol{m}(\boldsymbol{\phi}'_i)$)

    end

    Rank $\boldsymbol{c}_i, \boldsymbol{\nabla}_i$ by Δ_i

    Adapt CMA-ES parameters $\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{p}$ based on rankings of $\boldsymbol{c}_i$

    $\boldsymbol{\phi}^* \leftarrow \boldsymbol{\phi}^* + \eta \sum_{i=1}^{\lambda} w_i \boldsymbol{\nabla}_{\text{rank[i]}}$ `//` w_i `is part of` $\boldsymbol{p}$

    if *there is no change in* $\mathcal{A}$ **then**

      Restart CMA-ES with $\boldsymbol{\mu} = \mathbf{0}, \boldsymbol{\Sigma} = \sigma_g \boldsymbol{I}$

      Set $\boldsymbol{\phi}^*$ to a randomly selected elite from $\mathcal{A}$

    end

    $f(\boldsymbol{\phi}_q), \boldsymbol{m}(\boldsymbol{\phi}_q) \leftarrow evaluate(\boldsymbol{\phi}_q)$

    UPDATE_ARCHIVE($\mathcal{A}, \boldsymbol{\phi}_q, f(\boldsymbol{\phi}_q), \boldsymbol{m}(\boldsymbol{\phi}_q)$)

    Add experience from all calls to $evaluate$ into $\mathcal{B}$

    TRAIN_TD3($Q_{\boldsymbol{\theta}_1}, Q_{\boldsymbol{\theta}_2}, \pi_{\boldsymbol{\phi}_q}, Q_{\boldsymbol{\theta}'_1}, Q_{\boldsymbol{\theta}'_2}, \pi_{\boldsymbol{\phi}'_q}, \mathcal{B}$)

  end

B.2 HELPER METHODS FOR CMA-MEGA VARIANTS

Algorithm 2: Helper method 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, $f(\boldsymbol{\phi}) - f(\boldsymbol{\phi}_{\mathcal{E}})$)
  end

Algorithm 3: TD3 helper methods. Adapted from PGA-MAP-Elites (Nilsson and Cully, 2021) and TD3 (Fujimoto et al., 2018).

INITIALIZE_TD3:
  $\mathcal{B} \leftarrow$ initialize_replay_buffer()
  `// As done in the TD3 author implementation (Fujimoto et al., 2018), 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.`
      $\boldsymbol{\theta}'_i \leftarrow \tau \boldsymbol{\theta}_i + (1-\tau)\boldsymbol{\theta}'_i$
      $\boldsymbol{\phi}'_q \leftarrow \tau \boldsymbol{\phi}_q + (1-\tau)\boldsymbol{\phi}'_q$
    end
  end

Algorithm 4: Helper method for estimating objective and measure gradients with the gradient estimate from OpenAI-ES. This implementation differs from Eq. 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` $\boldsymbol{x}_i$`.`
  - $f(\boldsymbol{x}'_i), \boldsymbol{m}(\boldsymbol{x}'_i) \leftarrow evaluate(\boldsymbol{x}_i)$
- **end**
- **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
  - **end**
  - `// 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**
    - $\boldsymbol{\nabla}_{m_j}(\boldsymbol{\phi}) \leftarrow \boldsymbol{\nabla}$
  - **end**
- **end**
- **return** $\boldsymbol{\nabla}_f(\boldsymbol{\phi}), \boldsymbol{\nabla}_{\boldsymbol{m}}(\boldsymbol{\phi})$

C ALGORITHM HYPERPARAMETERS

Here we list parameters for each algorithm in our experiments. Refer to Sec. 5.1.2 for parameters of the neural network policy and the archive. All algorithms are allocated 1,000,000 evaluations total.

Table 1: 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-MAP-Elites, but it is much larger here 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 2: PGA-MAP-Elites 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: 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 4: MAP-Elites hyperparameters. We describe MAP-Elites in Sec. 3.2.1.

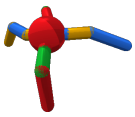
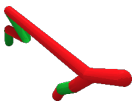
Parameter	Description	Value
N	Iterations = 1,000,000 / λ	10,000
λ	Batch size	100
σ	Gaussian noise standard deviation	0.02

Table 5: TD3 hyperparameters common to CMA-MEGA (TD3, ES) and PGA-MAP-Elites, which both train a TD3 instance. Furthermore, though we record the objective with $\gamma = 1$ (Sec. 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

D ENVIRONMENT DETAILS

Table 6: 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. 5.1.3). See Appendix D.1 for more info on the measures.

	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

D.1 MEASURES

The measures in QDGym are the proportions of time that each foot contacts the ground. In each environment, the feet are ordered as follows:

- **QD Ant:** front left foot, front right foot, back left foot, back right foot
- **QD Half-Cheetah:** front foot, back foot
- **QD Hopper:** single foot
- **QD Walker:** right foot, left foot

E FINAL METRICS

Tables 7-12 show the QD score (Sec. 5.1.3), QD score AUC (Sec. G.1.1), archive coverage (Sec. 5.1.3), best performance (Sec. 5.1.3), mean elite robustness (Sec. G.3), and runtime in hours (Sec. G.5) 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}.

Table 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 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

Table 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 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

Table 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 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

F FULL STATISTICAL ANALYSIS

To compare a metric such as QD score between two or more algorithms across all four QDGym environments, we performed a two-way ANOVA where environment and algorithm were the independent variables and the metric was the dependent variable. When there was a significant interaction effect (note that all of our analyses found significant interaction effects), we followed up this ANOVA with a simple main effects analysis in each environment. Finally, we ran pairwise comparisons (two-sided t-tests) to determine which algorithms had a significant difference on the metric. We applied Bonferroni corrections within each environment / simple main effect. For example, in Table 13 we compared CMA-MEGA (ES) with three algorithms in each environment, so we applied a Bonferroni correction with $n = 3$.

This section lists the ANOVA and pairwise comparison results for each of our analyses. We have **bolded** all significant p-values, where significance is determined at the $\alpha = 0.05$ threshold. For pairwise comparisons, some p-values are marked as "1" because the Bonferroni correction caused the p-value to exceed 1. p-values less than 0.001 have been marked as "**< 0.001**".

F.1 QD SCORE ANALYSIS (SEC. G.1)

To test the hypotheses we defined in Sec. 5.2, we performed a two-way ANOVA for QD scores. Since the ANOVA requires scores in all environments to have the same scale, we normalized the QD score in all environments by dividing by the maximum QD score, defined in Sec. G.1 as *grid cells* * (*max objective - min objective*). The results of the ANOVA were as follows:

- Interaction effect: $F(12, 80) = 16.82, \mathbf{p < 0.001}$
- Simple main effects:
  - QD Ant: $F(4, 80) = 23.87, \mathbf{p < 0.001}$
  - QD Half-Cheetah: $F(4, 80) = 44.15, \mathbf{p < 0.001}$
  - QD Hopper: $F(4, 80) = 57.35, \mathbf{p < 0.001}$
  - QD Walker: $F(4, 80) = 90.84, \mathbf{p < 0.001}$

Since the ANOVA showed a significant interaction effect and significant simple main effects, we performed pairwise comparisons for each hypothesis (Tables 13-15).

F.2 QD SCORE AUC ANALYSIS (SEC. G.1.1)

In this followup analysis, we hypothesized that PGA-MAP-Elites would have greater QD score AUC than CMA-MEGA (ES) and CMA-MEGA (TD3, ES). Thus, we performed a two-way ANOVA which compared QD score AUC for PGA-MAP-Elites, CMA-MEGA (ES), and CMA-MEGA (TD3, ES). As we did for QD score, we normalized QD score AUC by the maximum QD score. The ANOVA results were as follows:

- Interaction effect: $F(12, 80) = 17.55, \mathbf{p < 0.001}$
- Simple main effects:
  - QD Ant: $F(4, 80) = 31.77, \mathbf{p < 0.001}$
  - QD Half-Cheetah: $F(4, 80) = 89.38, \mathbf{p < 0.001}$
  - QD Hopper: $F(4, 80) = 82.34, \mathbf{p < 0.001}$
  - QD Walker: $F(4, 80) = 71.64, \mathbf{p < 0.001}$

As the interaction and simple main effects were significant, we performed pairwise comparisons (Table 16).

F.3 MEAN ELITE ROBUSTNESS ANALYSIS (SEC. G.3)

In this followup analysis, we hypothesized that MAP-Elites would have lower mean elite robustness than CMA-MEGA (ES) and CMA-MEGA (TD3, ES). Thus, we performed a two-way ANOVA which compared mean elite robustness for MAP-Elites, CMA-MEGA (ES), and CMA-MEGA

(TD3, ES). We normalized by the score range, i.e. *max objective - min objective*. The ANOVA results were as follows:

- Interaction effect: $F(12, 80) = 8.75, \mathbf{p < 0.001}$
- Simple main effects:
  - QD Ant: $F(4, 80) = 3.17, \mathbf{p = 0.018}$
  - QD Half-Cheetah: $F(4, 80) = 9.60, \mathbf{p < 0.001}$
  - QD Hopper: $F(4, 80) = 21.07, \mathbf{p < 0.001}$
  - QD Walker: $F(4, 80) = 3.70, \mathbf{p = 0.008}$

As the interaction and simple main effects were significant, we performed pairwise comparisons (Table 17).

Table 13: H1 - Comparing QD score between CMA-MEGA (ES) and baselines

Algorithm 1	Algorithm 2	QD Ant	QD Half-Cheetah	QD Hopper	QD Walker
CMA-MEGA (ES)	PGA-MAP-Elites	1	0.733	**0.003**	**< 0.001**
	ME-ES	**< 0.001**	**< 0.001**	0.841	**< 0.001**
	MAP-Elites	0.254	0.215	**0.007**	0.108

Table 14: H2 - Comparing QD score between CMA-MEGA (TD3, ES) and baselines

Algorithm 1	Algorithm 2	QD Ant	QD Half-Cheetah	QD Hopper	QD Walker
CMA-MEGA (TD3, ES)	PGA-MAP-Elites	0.093	1	0.726	1
	ME-ES	**< 0.001**	**< 0.001**	**< 0.001**	**< 0.001**
	MAP-Elites	1	**0.010**	1	**< 0.001**

Table 15: H3 - Comparing QD score between CMA-MEGA (ES) and CMA-MEGA (TD3, ES)

Algorithm 1	Algorithm 2	QD Ant	QD Half-Cheetah	QD Hopper	QD Walker
CMA-MEGA (ES)	CMA-MEGA (TD3, ES)	0.250	0.511	**0.006**	**< 0.001**

Table 16: Comparing QD score AUC between PGA-ME and CMA-MEGA variants

Algorithm 1	Algorithm 2	QD Ant	QD Half-Cheetah	QD Hopper	QD Walker
PGA-MAP-Elites	CMA-MEGA (ES)	0.734	0.255	**< 0.001**	**< 0.001**
	CMA-MEGA (TD3, ES)	**0.020**	0.111	**0.003**	1

Table 17: Comparing mean elite robustness between MAP-Elites and CMA-MEGA variants

Algorithm 1	Algorithm 2	QD Ant	QD Half-Cheetah	QD Hopper	QD Walker
MAP-Elites	CMA-MEGA (ES)	**< 0.001**	**< 0.001**	**0.030**	**0.003**
	CMA-MEGA (TD3, ES)	**< 0.001**	**< 0.001**	**0.013**	**< 0.001**

G DISCUSSION

G.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*) (see Table 6 for these quantities).

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 (Appendix F). Our results are as follows:

- **H1:** There is no significant difference in QD score between CMA-MEGA (ES) and PGA-MAP-Elites in QD Ant and QD Half-Cheetah, but in QD Hopper and QD Walker, CMA-MEGA (ES) attains significantly lower QD score than PGA-MAP-Elites. 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.
- **H2:** In all environments, there is no significant difference in QD score between CMA-MEGA (TD3, ES) and PGA-MAP-Elites. 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 QD Walker, but there is 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.

G.1.1 PGA-MAP-ELITES AND OBJECTIVE-MEASURE SPACE EXPLORATION

Of the CMA-MEGA variants, CMA-MEGA (TD3, ES) performed the closest to PGA-MAP-Elites, with no significant QD score difference in any environment. This result differs from prior work (Fontaine and Nikolaidis, 2021) in QD benchmark domains, where CMA-MEGA outperformed OG-MAP-Elites, a baseline DQD algorithm inspired by PGA-MAP-Elites.

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" (Fontaine et al., 2020). 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 traversing measure space and filling the archive. In contrast, as discussed in Sec. 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. This may even happen within the first 100k evaluations, as in QD Half-Cheetah (Fig. 3). Hence, high QD score in QDGym comes from optimizing solutions after filling the archive.

Since CMA-MEGA adapts its sampling distribution, it performs the exploration necessary to succeed in the linear projection domain, while OG-MAP-Elites remains "stuck" in the center of the measure space. However, this exploration capability does not increase QD score in QDGym. Instead, QDGym is more appropriate for an algorithm which rigorously optimizes the objective. PGA-MAP-Elites does exactly this: every iteration, it increases the objective value of half of its generated solutions by optimizing them with respect to a TD3 critic. Though CMA-MEGA (TD3, ES) also trains a TD3 critic, it does not perform this additional optimization. Naturally, this leads to a possible extension in which solutions sampled by CMA-MEGA (TD3, ES) are optimized with respect to its TD3 critic before being evaluated in the environment.

G.1.2 PGA-MAP-ELITES AND OPTIMIZATION EFFICIENCY

While there was no significant difference in the *final* QD scores of CMA-MEGA (TD3, ES) and PGA-MAP-Elites, CMA-MEGA (TD3, ES) was less *efficient* than PGA-MAP-Elites in some environments. For instance, in QD Hopper, PGA-MAP-Elites 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:

$$\text{QD score AUC} = \sum_{i=1}^{N} (\lambda * \text{QD score at iteration } i) \tag{4}$$

After computing QD score AUC, we ran statistical analysis similar to Sec. G.1 and found CMA-MEGA (TD3, ES) had significantly lower QD score AUC than PGA-MAP-Elites 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-MAP-Elites in all tasks, it was less efficient at achieving those scores in QD Ant and QD Hopper.

G.2 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 (Appendix C) 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 variants also evaluate 200 solutions per iteration, they insert 100 of these into the archive. This totals to 500k insertions per run, allowing the variants to gradually improve archive cells.

G.3 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, when we visualized solutions found by MAP-Elites, their performance was lower than the performance recorded in the archive. The best MAP-Elites solution in QD Hopper hopped forward a few steps and fell down, despite recording an excellent performance of 2,648.31 (see supplemental videos).

One explanation for this behavior is that since we only evaluate solutions for one episode before inserting into the archive, a solution with noisy performance may be inserted because of a single high-performing episode, even if it performs poorly on average. Prior work (Nilsson and Cully, 2021) has also encountered this issue when running MAP-Elites with a directional variation operator (Vassiliades and Mouret, 2018) in QDGym, and has suggested measuring *robustness* as a proxy for how much noise is present in an archive's solutions. Robustness is defined as the difference between the mean performance of the solution over n episodes (we use $n = 10$) and the performance recorded in the archive. The larger (more negative) this difference, the more noisy and less robust the solution.

To compare the robustness of the solutions output by the CMA-MEGA variants and MAP-Elites, we computed *mean elite robustness*, the average robustness of all elites in each experiment's final archive. We then ran statistical analysis similar to Sec. G.1. In all environments, both CMA-MEGA (ES) and CMA-MEGA (TD3, ES) had significantly higher mean elite robustness than MAP-Elites (Appendix E & F). Overall, though MAP-Elites achieves high QD score, its solutions are less robust.

G.4 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 (Pagliuca et al., 2020), these signals led PPO (Schulman et al., 2017) 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-MAP-Elites and CMA-MEGA (TD3, ES).

Prior work (Pagliuca et al., 2020) 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-MAP-Elites and CMA-MEGA (TD3, ES).

G.5 Computational effort

Though CMA-MEGA (TD3, ES) and PGA-MAP-Elites perform best overall, they rely on specialized hardware (a GPU) and require the most computation. As shown in Table 12, Appendix E, the TD3 training in these algorithms leads to long runtimes. When runtime is dominated by the algorithm itself (as opposed to solution evaluations), CMA-MEGA (ES) offers a viable alternative that may achieve reasonable performance.

H ARCHIVE VISUALIZATIONS

We visualize "median" archives in Fig. 4 and Fig. 5. To determine these median archives, we selected the trial which achieved the median QD score out of the 5 trials of each algorithm in each environment. Fig. 4 visualizes heatmaps of median archives in QD Half-Cheetah and QD Walker, while Fig. 5 shows the distribution (histogram) of objective values for median archives in all environments. Refer to the supplemental material for videos of how these figures develop across iterations.

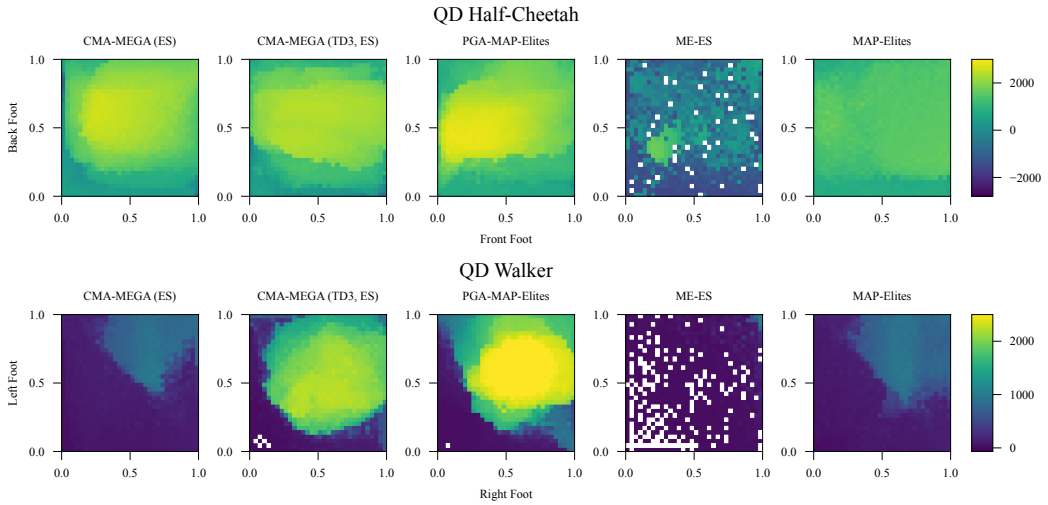


Figure 4: Archive heatmaps from the median trial (in terms of QD score) of each algorithm in QD Half-Cheetah and QD Walker. The colorbar for each environment ranges from the minimum to maximum objective stated in Table 6. The archive in both environments is a 32×32 grid. Currently, we are unable to plot heatmaps for QD Ant and QD Hopper because their archives are not 2D. Refer to the supplemental material for a video of how these archives develop across iterations.

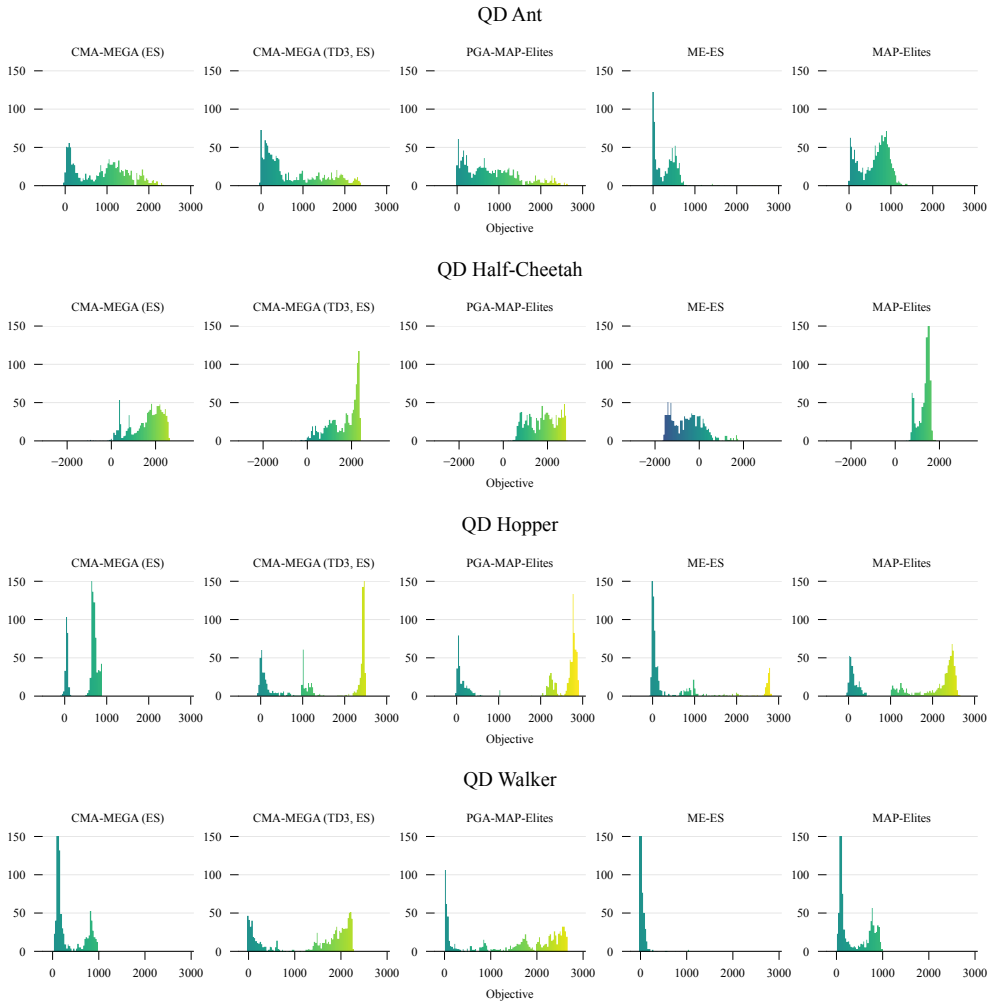


Figure 5: Distribution (histogram) of objective values in archives from the median trial (in terms of QD score) of each algorithm in each environment. In each plot, the x-axis is bounded on the left by the minimum objective and on the right by the maximum objective plus 400, as some solutions exceed the maximum objective in Table 6. Note that in some plots, the number of items overflows the y-axis bounds (e.g. ME-ES in QD Walker). Refer to the supplemental material for a video of how these distributions develop across iterations.