



HAL
open science

Scaling MAP-Elites to Deep Neuroevolution

Cédric Colas, Vashisht Madhavan, Joost Huizinga, Jeff Clune

► **To cite this version:**

Cédric Colas, Vashisht Madhavan, Joost Huizinga, Jeff Clune. Scaling MAP-Elites to Deep Neuroevolution. GECCO 2020 - Genetic and Evolutionary Computation Conference, Jul 2020, Cancun / Virtual, Mexico. 10.1145/3377930.3390217 . hal-03099878

HAL Id: hal-03099878

<https://inria.hal.science/hal-03099878>

Submitted on 6 Jan 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Scaling MAP-Elites to Deep Neuroevolution

Cédric Colas*
INRIA

Vashisht Madhavan
Element Inc.

Joost Huizinga
Uber AI Labs

Jeff Clune
OpenAI

ABSTRACT

Quality-Diversity (QD) algorithms, and MAP-Elites (ME) in particular, have proven very useful for a broad range of applications including enabling real robots to recover quickly from joint damage, solving strongly deceptive maze tasks or evolving robot morphologies to discover new gaits. However, present implementations of ME and other QD algorithms seem to be limited to low-dimensional controllers with far fewer parameters than modern deep neural network models. In this paper, we propose to leverage the efficiency of Evolution Strategies (ES) to scale MAP-Elites to high-dimensional controllers parameterized by large neural networks. We design and evaluate a new hybrid algorithm called MAP-Elites with Evolution Strategies (ME-ES) for post-damage recovery in a difficult high-dimensional control task where traditional ME fails. Additionally, we show that ME-ES performs efficient exploration, on par with state-of-the-art exploration algorithms in high-dimensional control tasks with strongly *deceptive* rewards.

CCS CONCEPTS

• **Computing methodologies** → **Evolutionary robotics**;

KEYWORDS

Quality-Diversity, Evolution Strategies, Map-Elites, Exploration

ACM Reference Format:

Cédric Colas, Joost Huizinga, Vashisht Madhavan, and Jeff Clune. 2020. Scaling MAP-Elites to Deep Neuroevolution. In *Genetic and Evolutionary Computation Conference (GECCO '20)*, July 8–12, 2020, Cancún, Mexico. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3377930.3390217>

1 INTRODUCTION

The path to success is not always a straight line. This popular saying describes, in simple terms, a key problem of non-convex optimization. When optimizing a model for a non-convex objective function, an algorithm that greedily follows the gradient of the objective might get stuck in local optima. One can think of an agent in a maze with numerous walls. An algorithm that minimizes the

distance between the position of the agent and the maze center will surely lead to the agent getting stuck in a corner. Based on this observation, Lehman and Stanley [30] proposed a thought-provoking idea: ignore the objective and optimize for novelty instead. Novelty search (NS) continually generates new behaviors without considering any objective and, as such, is not subject to the local optima encountered by algorithms following fixed objectives. In cases like our maze example, NS can lead to better solutions than objective-driven optimization [30].

Despite the successes of NS, objectives still convey useful information for solving tasks. As the space of possible behaviors increases in size, NS can endlessly generate novel outcomes, few of which may be relevant to the task at hand. Quality-Diversity (QD) algorithms address this issue by searching for a collection of solutions that is both diverse and high-performing [14, 31, 34]. MAP-Elites, in particular, was used to generate diverse behavioral repertoires of walking gaits on simulated and physical robots, which enabled them to recover quickly from joint damage [13].

In QD algorithms like MAP-Elites or NSLC, a Genetic Algorithm (GA) is often used as the underlying optimization algorithm. A candidate controller is selected to be mutated and the resulting controller is evaluated in the environment, leading to a performance measure (fitness) and a behavioral characterization (low-dimensional representation of the agent’s behavior). QD algorithms usually curate some form of a *behavioral repertoire*, a collection of high-performing and/or diverse controllers experienced in the past (sometimes called *archive* [14, 31] or *behavioral map* [34]). Each newly generated controller can thus be added to the behavioral repertoire if it meets some algorithm-dependent conditions. From a parallel line of work, Intrinsically Motivated Goal Exploration Processes (IMGEP) also curate behavioral repertoires and are often based on GA-like optimizations [3, 18]. Agents are able to set their own goals in the behavioral space, and try to reach them by combining controllers that reached behavioral characterizations (BCs) close to these goals. Uniform goal selection here triggers a novelty effect where controllers reaching sparse areas are used more often, while learning-progress-based sampling implements a form of QD where quality is defined as the ability to reliably achieve goals [3].

Thus far, the most successful demonstrations of QD algorithms have been on robotics problems with relatively simple, low-dimensional controllers [14, 30, 31, 34]. Modern robot controllers such as ones based on neural networks can have millions of parameters, which makes them difficult –though not impossible– to optimize with Evolutionary Algorithms [47]. Such large controllers are usually trained through Deep Reinforcement Learning (DRL), where an agent learns to perform a sequence of actions in an environment so as to maximize some notion of cumulative reward [48]. DRL is concerned with training deep neural networks (DNNs), typically

*Corresponding author: cedric.colas@inria.fr. Work conducted during an internship at Uber AI Labs. All authors were affiliated to Uber AI Labs at the time.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions.acm.org).

GECCO '20, July 8–12, 2020, Cancún, Mexico

© 2020 Association for Computing Machinery.

ACM ISBN 978-1-4503-7128-5/20/07...\$15.00

<https://doi.org/10.1145/3377930.3390217>

via stochastic gradient descent (SGD) to facilitate learning in robot control problems. Unlike in supervised learning, training data in DRL is generated by having the agent interact with the environment. If the agent greedily takes actions to maximize reward –a phenomenon known as *exploitation*– it may run into local optima and fail to discover alternate strategies with larger payoffs. To avoid this, RL algorithms also need *exploration*. RL algorithms usually explore in the action space, adding random noise on a controller’s selected actions (ϵ -greedy) [20, 32, 33]. More directed exploration techniques endow the agent with *intrinsic motivations* [4, 37, 44]. Most of the time, the reward is augmented with an *exploration bonus*, driving the agent to optimize for proxies of *uncertainty* such as novelty [5], prediction error [9, 38], model-disagreement [46], surprise [1] or expected information gain [26].

In contrast with QD algorithms however, DRL algorithms are usually concerned with training a single controller to solve tasks [20, 32, 33, 38]. The agent thus needs to rely on a single controller to adapt to new environments [36, 39], tasks [16], or adversarial attacks [22]. Kume et al. [28] outlines perhaps the first RL algorithm to form a behavioral repertoire by training multiple policies. Optimizing for performance, this algorithm relies on the instabilities of the underlying learning algorithm (Deep Deterministic Policy Gradient [32]) to generate a diversity of behaviors that will be collected in a *behavioral map*. Although the performance optimization of this QD algorithm leverages DRL, exploration remains incidental.

In recent years, Deep Neuroevolution has emerged as a powerful competitor to SGD for training DNNs. Salimans et al. [43], in particular, presents a scalable version of the Evolution Strategies (ES) algorithm, achieving performances comparable to state-of-the-art DRL algorithms on high-dimensional control tasks like Mujoco [8] and the Atari suite [6]. Similarly, Genetic Algorithms (GA) were also shown to be capable of training DNN controllers for the Atari suite, but failed to do so on the Mujoco suite [47]. ES, in contrast with GA, combines information from many perturbed versions of the parent controller to generate a new one. Doing so allows the computation of gradient estimates, leading to efficient optimization in high-dimensional parameter spaces like DNNs [43]. Recent implementations also utilize the resources of computing clusters by parallelizing the controller evaluations, making ES algorithms competitive with DRL in terms of training time [43]. Since Such et al. [47] showed that even powerful, modern GAs using considerable amounts of computation could not solve continuous control tasks like those from the Mujoco suite, we propose to unlock QD for high-dimensional control tasks via a novel QD-ES hybrid algorithm. Doing so, we aim to benefit from the exploration abilities and resulting behavioral repertoires of QD algorithms, while leveraging the ability of ES to optimize large models.

NS-ES made a first step in this direction by combining NS with ES: replacing the performance objective of ES with a novelty objective [12]. Two variants were proposed to incorporate the performance objective: NSR-ES, which mixes performance and novelty objectives evenly and NSRA-ES, which adaptively tunes the ratio between the two. These methods –like most RL exploration methods– use a mixture of exploitation and exploration objectives as a way to deal with the *exploitation-exploration tradeoff*. While this might work when the two objectives are somewhat aligned, it may be

Table 1: Classification of related algorithms

	Gradient-Based	Mutation-Based
Pure Exploitation	ES	GA
Pure Exploration	NS-ES	NS
Exploration & Exploitation	Coupled: NSR-ES, NSRA-ES Decoupled: ME-ES	ME-GA

inefficient when they are not [40]. Several works have started to investigate this question and some propose to disentangle exploration and exploitation into distinct phases [7, 11, 52]. QD presents a natural way of *decoupling* the optimization of exploitation (quality) and exploration (diversity) by looking for high-performing solutions in local niches of the behavioral space, leading to local competition between solutions instead of a global competition [14, 31, 34].

Contributions. In this work, we present ME-ES, a version of the powerful QD algorithm MAP-Elites that scales to hard, high-dimensional control tasks by leveraging ES. Unlike NS-ES and its variants, which optimize a small population of DNN controllers (e.g. 5) [12], ME-ES builds a large repertoire of diverse controllers. Having a set of diverse, high-performing behaviors not only enables efficient exploration but also robustness to perturbations (either in the environment itself or in the agent’s abilities). In behavioral repertoires, the burden of being robust to perturbations can be shared between different specialized controllers, one of which can solve the new problem at hand. Algorithms that train a single controller, however, need this controller to be robust to all potential perturbations. We present two applications of ME-ES. The first is damage recovery in a high-dimensional control task: after building a repertoire of behaviors, the agent is damaged (e.g. disabled joints) and must adapt to succeed. We show that ME-ES can discover adaptive behaviors that perform well despite the damage, while a previous implementation of MAP-Elites based on GA fails. The second application is exploration in environments with strongly deceptive rewards. We show that agents trained with ME-ES perform on par with state-of-the-art exploration algorithms (NS-ES and variants). Table 1 presents a classification of ME-ES and related algorithms presented in this paper along three dimensions: 1) whether they use pure exploration, pure exploitation or both, 2) whether they rely on gradient-based or mutation-based learning algorithms and 3) whether they couple or decouple the trade-off between exploration and exploitation (if applicable). As ES computes *natural gradient* estimations through Monte-Carlo approximations [50], we refer to ES-powered methods as *gradient-based*.

2 BACKGROUND

2.1 MAP-Elites

MAP-Elites was first presented in Mouret and Clune [34]. In addition to an objective function –or *fitness* function (F)– which measures the performance of an agent, MAP-Elites assumes the definition of a behavioral characterization (BC) mapping the state-action

trajectory of an agent in its environment to a low-dimensional embedding lying in a *behavioral space*. MAP-Elites keeps track of an archive of individuals (controllers) tried in the past along with their associated fitness values and behavioral characterizations. The aim is to curate a repertoire of behaviorally diverse and high-performing agents. To this end, the behavioral space is discretized into *cells* representing behavioral *niches*, where each niche maintains the highest performing individual whose behavioral characterization falls into its cell. Individuals in each niche optimize for the fitness objective, yet implicitly have a pressure for diversity, as they are driven towards empty and under-optimized cells to avoid the selection pressure of cells with highly optimized individuals.

After initializing the archive with a few randomly initialized controllers, MAP-Elites repeats the following steps:

- (1) Select a populated cell at random,
- (2) Mutate the cell’s controller (GA) to obtain a new controller,
- (3) Gather a trajectory of agent-environment interactions with the new controller and obtain its fitness and *BC*,
- (4) Update the archive: add the controller to the cell where the *BC* falls if *Rule 1*) the cell is empty or *Rule 2*) the fitness is higher than that of the controller currently in the cell.

The two rules guiding additions to the archive implement a decoupling between exploitation (quality) and exploration (diversity). *Rule 1* implements exploration, as it ensures the preservation of controllers that exhibit novel behavior (i.e. mapped to empty cells). *Rule 2* implements exploitation, as it enforces local competition and retains only the highest performing solution in each behavioral niche. In this manner, the exploration and exploitation objectives cannot contradict each other. We call the traditional implementation of MAP-Elites based on GA *ME-GA* while *MAP-Elites* encompasses both ME-GA and ME-ES variants.

2.2 Evolution Strategies

Evolution Strategies (ES) is a class of black box optimization algorithms inspired by natural evolution [2, 42]. For each generation, an initial parameter vector (the *parent*), is mutated to generate a population of parameter vectors (the *offspring*). The *fitness* of each resultant offspring is evaluated and the parameters are combined such that individuals with high fitness have higher influence than others. As this process is repeated, the population tends towards regions of the parameter space with higher fitness, until a convergence point is reached.

We use a version of ES introduced in Salimans et al. [43], which itself belongs to the subcategory of Natural Evolution Strategies (NES) [45, 50]. In NES, the population of parameter vectors θ is represented by a distribution $p_\psi(\theta)$, parameterized by ψ . Given an objective function F , NES algorithms optimize the expected objective value $\mathbb{E}_{\theta \sim p_\psi} F(\theta)$ using stochastic gradient ascent. Recently, Salimans et al. [43] proposed a version of the NES algorithm able to scale to the optimization of high-dimensional parameters ($\approx 10^5$). In their work, they address the RL problem, for which θ refers to the parameters of a controller while the fitness function is the reward obtained by the corresponding controller over an episode of environment interactions. Given the parent controller parameters θ , the offspring population ($\theta_i \forall i \in [1..n]$) is sampled from the isotropic multivariate Gaussian distribution $\theta_i \sim \mathcal{N}(\theta, \sigma^2 I)$ with

fixed variance σ^2 . As in REINFORCE [51], θ is updated according to the following gradient approximation:

$$\nabla_\psi \mathbb{E}_{\theta \sim p_\psi} [F(\theta)] \approx \frac{1}{n} \sum_{i=1}^n F(\theta_i) \nabla_\psi \log p_\psi(\theta),$$

where n is the offspring population size, usually large to compensate for the high variance of this estimate. In practice, any θ_i can be decomposed as $\theta_i = \theta + \sigma \epsilon_i$, and the gradient is estimated by:

$$\nabla_\psi \mathbb{E}_{\epsilon \sim \mathcal{N}(0, I)} [F(\theta + \sigma \epsilon)] \approx \frac{1}{n\sigma} \sum_{i=1}^n F(\theta_i) \epsilon_i.$$

As in Salimans et al. [43], we use virtual batch normalization of the controller’s inputs and rank-normalize the fitness $F(\theta^i)$. The version of NES used in this paper is strictly equivalent to the one proposed in Salimans et al. [43]. We simply refer to it as ES hereafter.

3 METHODS

3.1 ME-ES

The ME-ES algorithm reuses the founding principles of MAP-Elites leading to damage robustness and efficient exploration, while leveraging the optimization performance of ES. ME-ES curates a *behavioral map (BM)*, an archive of neural network controllers parameterized by θ . Every $n_{\text{optim_gens}}$ generations, a populated cell and its associated controller θ_{cell} are sampled from the archive. This controller is then copied to θ_g , which is subsequently evolved for $n_{\text{optim_gens}}$ generations using ES. At each generation, offspring parameters $\theta'_g \sim \mathcal{N}(\theta_g, \sigma^2 I)$ are only used to compute an update for θ_g . We only consider θ_g for addition to the *BM* to ensure robust evaluations (30 episodes) and a fair comparison with competing algorithms such as NS-ES. Algorithm 1 provides a detailed outline of the way ME-ES combines principles from MAP-Elites and ES.

Algorithm 1 ME-ES Algorithm

- 1: **Input:** n_{gens} , pop_size , σ , $n_{\text{optim_gens}}$, empty behavioral map *BM*, n_{eval}
 - 2: **Initialize:** $BM \leftarrow (\theta_0 \sim \text{init}_{\text{normc}}(), F(\theta_0), BC(\theta_0))$
 - 3: **for** $g = [0, n_{\text{gens}}]$ **do**
 - 4: **if** $g \% n_{\text{optim_gens}} == 0$ **then**
 - 5: $\text{mode} \leftarrow \text{explore_or_exploit}()$
 - 6: **if** $\text{mode} == \text{'explore'}$ **then**
 - 7: $\theta_{\text{cell}} \leftarrow \text{sample_explore_cell}()$
 - 8: **else if** $\text{mode} == \text{'exploit'}$ **then**
 - 9: $\theta_{\text{cell}} \leftarrow \text{sample_exploit_cell}()$
 - 10: $\theta_g \leftarrow \theta_{\text{cell}}$
 - 11: $\theta_g \leftarrow \text{ES_optim}(\theta_g, \text{pop_size}, \sigma, \text{objective}=\text{mode})$
 - 12: $F(\theta_g), BC(\theta_g) \leftarrow \text{Evaluate}(\theta_g, n_{\text{eval}})$
 - 13: $\text{Update_BM}(\theta_g, F(\theta_g), BC(\theta_g))$
-

ME-ES variants. We define three variants of ME-ES that differ in the objective optimized by ES:

- In ME-ES EXPLOIT, the objective is the fitness function F . $F(\theta)$ is computed by running the agent’s controller in the environment for one episode (*directed exploitation*).

- In ME-ES EXPLORE, the objective is the *novelty* function $N(\theta, k)$, which we define as the average Euclidean distance between a controller’s *BC* and its k nearest neighbors in an *archive* storing all previous θ_g , regardless of their addition to the *BM* (one per generation). Because the novelty objective explicitly incentivizes agents to explore their environment, we call it a *directed exploration* objective.
- Finally, ME-ES EXPLORE-EXPLOIT alternates between both objectives, thus implementing a decoupled exploration and exploitation procedure. Because it explicitly optimizes for both objectives, ME-ES EXPLORE-EXPLOIT conducts both *directed exploration* and *directed exploitation*.

Note that ME-ES EXPLORE performs undirected exploitation; while the ES steps do not directly optimize for fitness, performance measures are still used to update the *BM* (Rule 2 of the map updates). In the same way, ME-ES EXPLOIT also performs undirected exploration, as the *BM* is updated with novel controllers (Rule 1 of map updates). Like ME-GA, all versions of ME-ES thus perform forms of exploration and exploitation. Only optimization steps with ES, however, enable agents to perform the *directed* version of both exploitation and exploration.

Cell sampling. The number of generations performed by ME-ES is typically orders of magnitude lower than MAP-Elites, as each generation of ME-ES involves a greater number of episodes of environment interaction (10^4 instead of 1 for ME-GA). Thus, the sampling of the cell from which to initiate the next n_{optim_gens} generations is crucial (see Algorithm 1). Here, we move away from the cell selection via uniform sampling used in MAP-Elites, towards biased cell sampling. We propose two distinct strategies. For exploitation steps, we select from cells with high fitness controllers, under the assumption that high fitness cells may lead to even higher fitness cells. For exploration steps, we select from cells with high novelty scores. By definition, novel controllers are in under-explored areas of the search space and mutating these controllers should lead to novel areas of the behavior space [30, 31]. In practice, as the number of cells populated increases with each generation, the bias towards selecting cells with higher performance or novelty score diminishes. For instance, if cells are selected proportional to their novelty, the best controller of two that have novelty 2 and 1 will have a 2/3 chance of being selected, while the same controller in a group of 3 controllers with novelties [2, 1, 1] will only have probability 1/2 of being selected. To fight this phenomenon, we propose to restrict the novelty-proportional selection to the five most novel cells. Similarly, for exploitation steps, we sample either uniformly from the two highest fitness cells ($p = 0.5$), which promotes exploitation from the current best cells, or uniformly from the two highest fitness cells from the last five updated ($p = 0.5$), which promotes exploitation from newly discovered cells. We considered sampling cells with high performance progress, but this often led to the selection of sub-optimal controllers, as starting to move forward from a controller going backward is one of the best way to make progress.

Hyperparameters. We use fully connected neural network controllers with two hidden layers of size 256, *tanh* non-linearities, Xavier initialization [23], an Adam optimizer [27] with learning rate 0.01 and a l2-coefficient of 0.005 for regularization. We run

ES for $n_{optim_gens} = 10$ consecutive generations with a population size of $n = 10^4$ and a noise parameter $\sigma = 0.02$. After each step, we evaluate the average fitness $F(\theta_g)$, average behavioral characterization $BC(\theta_g)$ and associated novelty score $N(\theta_g)$ of θ_g over $n_{eval} = 30$ episodes. Novelty is computed as the average distance between the controller’s *BC* and its $k = 10$ nearest neighbors in the archive of all past *BC*s, regardless of whether they were added to the *BM*. The code and environment are provided at <https://github.com/uber-research/Map-Elites-Evolutionary>.

3.2 Damage Recovery

The Intelligent Trial and Error (IT&E) algorithm integrates the evolution of a behavioral map using MAP-Elites and a subsequent procedure enabling agents to recover from damage by searching the *BM* for efficient recovery controllers [13]. Here, we reuse this setting to compare the traditional implementation of MAP-Elites based on GA and our proposed ME-ES variants, see Fig. 1. Once the behavioral map has been filled by ME-GA or ME-ES, we can use it for damage adaptation (e.g. loss of control for one or several joints). As adaptation procedure, we use the map-based Bayesian optimization algorithm (M-BOA), part of the original IT&E [13]. Bayesian optimization is especially suitable to find the maximum of an unknown objective function for which it is costly to obtain samples. We define the objective function as $\mathcal{F} : BC \rightarrow F$. M-BOA initializes a model of \mathcal{F} using the behavioral map filled by MAP-Elites and updates the model using Gaussian Process (GP) regression. Data acquisition boils down to the evaluation of a controller θ in the perturbed environment, providing a new pair $(BC(\theta), F(\theta))$ to update the model \mathcal{F} . We follow the implementation of Cully et al. [13] and use a Mat rn kernel function with parameters $\nu = 5/2$ and $\rho = 0.03$ as well as hyperparameters $\kappa = 0.3$ and $\sigma_{noise}^2 = 0.01$ as Upper Confidence Bound exploration parameter and noise prior respectively (see code).

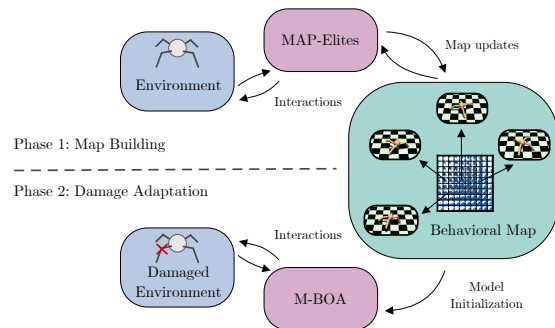


Figure 1: Building repertoires for damage adaptation. Phase 1: MAP-Elites builds a repertoire of diverse and high-performing behaviors. Phase 2: M-BOA builds a model of the objective function under the perturbed conditions to find a controller robust to the damage.

3.3 Baselines and Controls

In this paper, we test ME-ES in two applications. The first is the construction of behavioral repertoires for *damage adaptation* and the

second is exploration in deceptive reward environments (*deep exploration*). The two applications have different baselines and control treatments, which are described in the following sections.

3.3.1 MAP-Elites with Genetic Algorithm. We use the traditional implementation of MAP-Elites based on GA (ME-GA) as a baseline in the *damage adaptation* experiments in order to highlight the lift from using ES instead of GA to power MAP-Elites in high-dimensional control tasks. To ensure a fair comparison, it is important to keep the number of episodes constant between ME-GA and ME-ES. Here, ME-ES requires 10,030 episodes for each new controller (10^4 evaluations for the offsprings and 30 evaluations of the final controller to get a robust performance estimate). As such, we enable ME-GA to add up to 334 new controllers per generation, each being evaluated 30 times.

We do not use ME-GA as a baseline in the *deep exploration* experiments because the task builds on the Humanoid-v1 environment where Such et al. [47] demonstrated that GAs were not competitive with ES, even with 150x more environment interactions.

3.3.2 Novelty Search and Evolution Strategies. We compare ME-ES to NS-ES and its variants [12] for the *deep exploration* experiments. Since these baselines do not build behavioral repertoires, they are not used for the *damage adaptation* application. NS-ES replaces the performance objective of ES by the same novelty objective as ME-ES EXPLORE. Instead of a behavioral map, NS-ES and its variants keep a population of 5 parent controllers optimized in parallel via ES. NS-ES never uses fitness for optimization and thus is a pure exploration algorithm. We also compare against two variants of NS-ES that include a fitness objective: (1) NSR-ES optimizes the average of the fitness and novelty objectives, while (2) NSRA-ES implements an adaptive mixture where the mixing weight is updated during learning. For NSRA-ES, the mixing weight leans towards exploration (novelty objective) when the best fitness stagnates and shifts back towards exploitation (fitness objective) when new behaviors are discovered that lead to higher performance.

The weight adaptation strategy of NSRA-ES was modified in this paper. In the original work, the weight starts at 1 (pure exploitation), decreases by 0.05 every 50 generations if fitness does not improve and increases by 0.05 at every improvement. In practice, we found this cadence too slow to adapt properly, as the mixing weight can only go to 0 after 1000 generations in the best case. Thus, we increase the update frequency to every 20 generations and remove the bias towards exploitation at the start by setting the mixing weight to 0.5 initially.

4 EXPERIMENTS

Our two experiments are presented in the next sections. Recall that controllers are implemented by fully-connected neural networks with 2 hidden layers of size 256 (about 10^5 parameters). This results in search spaces that have orders of magnitude more dimensions than those used in traditional QD studies [14, 30, 31, 34]. For each treatment, we report the mean and standard deviations across 5 seeds, except for ME-GA, for which we present 3 seeds. ME-GA is about 3 times slower to run (> 3 days) and does not manage to learn recovery controllers in the *damage adaptation* task.

4.1 Building Repertoires for Damage Adaptation

We compare the quality of the *BM* generated by the different versions of MAP-Elites for damage recovery. In the first phase we build a behavioral map with each of the MAP-Elites algorithms and in the second phase we use the behavioral map created in phase 1 to help the agent recover from damage to its joints via M-BOA (Fig. 1). The damage applied to joints, indexed by \mathcal{J}_i , include:

- One joint cannot be controlled ($\mathcal{J}_{i \in [0..7]}$).
- One full leg cannot be controlled ($\mathcal{J}_{0,1}, \mathcal{J}_{2,3}, \mathcal{J}_{4,5}, \mathcal{J}_{6,7}$).

4.1.1 Domain. We evolve a repertoire of walking gaits in the Mujoco Ant-v2 environment [8]. The *BC* is a 4-D vector, where each dimension represents the proportion of episode steps where each leg is in contact with the floor ($[0, 1]^4$). This behavioral space is discretized into 10 bins along each dimension, for a total of 10^4 cells. The fitness is a mixture of the current x position and a cost on the torque for each joint. This incentivizes the agent to move as far along the x -axis as possible, without exerting too much energy.

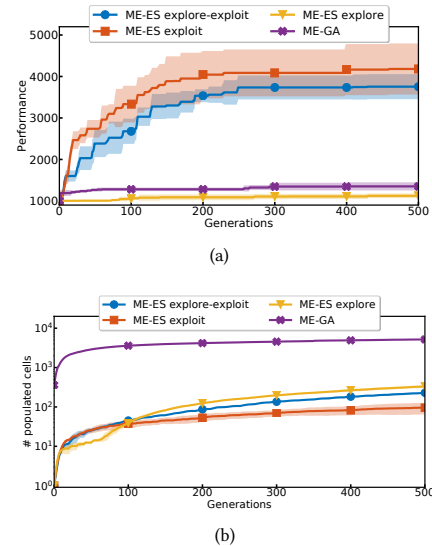


Figure 2: Ant behavioral map. a: Map performance (best performance in map). b: Map coverage (# populated cells), log scale on y-axis.

4.1.2 Results - Map Building. Fig. 2a shows the evolution of the map's best performance and 2b shows the map's coverage. As ME-GA adds up to 334 times more controllers to the map per generation, its coverage is orders of magnitude higher than ME-ES variants. However, the best performance found in the map is quite low for ME-GA. ME-ES EXPLORE also shows poor performances across its map. In contrast, ME-ES EXPLOIT and ME-ES EXPLORE-EXPLOIT show high performance, on par with Twin-Delayed Actor Critic (≈ 4200) [20] but below Soft-Actor Critic (≈ 6000) [24], two state-of-the-art DRL algorithms for this task.

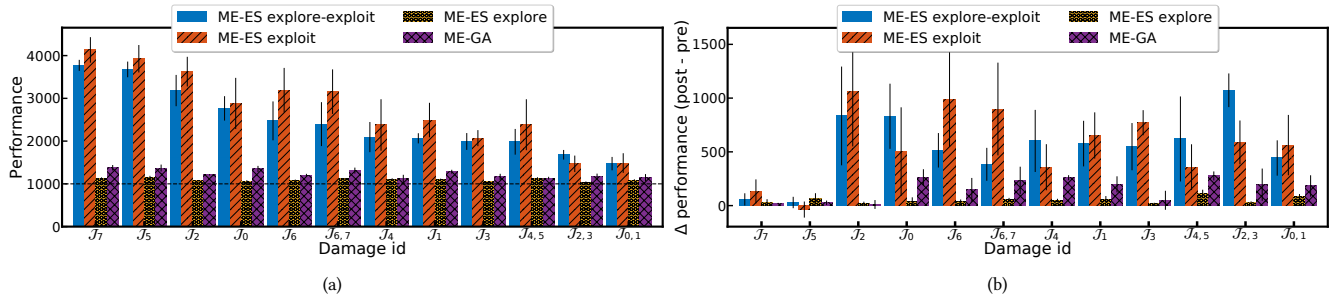


Figure 3: Damage adaptation. a: Performance of the best recovery controller. The joint damage is represented as J_i where i is the identifier of the broken joint or joints ($i \in [1..8]$). Dashed line represent the score of an agent that does not move forward but does not fall. b: Difference in performance, post- and pre-behavioral adaptation.

4.1.3 Results - Damage Adaptation. M-BOA is run on the behavioral maps from each experiment. Recovery controllers extracted from ME-GA and ME-ES EXPLORE maps do not achieve high scores (Fig. 3a). The scores are around 1000, which correspond to a motionless Ant (not moving forward, but not falling either). In contrast, ME-ES EXPLORE-EXPLOIT and ME-ES EXPLOIT are able to recover from joint damage and the ant is able to move in all damage scenarios (Fig. 3a). In Fig. 3b, we highlight the difference in performance before and after adaptation. Pre-adaptation performance is computed by evaluating the highest performing pre-damage controller in each of the post-damage environments. The post-adaptation performance is computed by evaluating the recovery controller in each of the damaged joint environments. In most cases, ME-ES EXPLORE-EXPLOIT and ME-ES EXPLOIT are able to significantly improve upon the pre-damage controller (Fig. 3b). In both figures, averages and standard errors are computed over different runs of MAP-Elites + M-BOA (different seeds). The performance of a given recovery controller is averaged over 30 episodes.

4.1.4 Interpretation. ME-GA does not scale to the high-dimensional Ant-v2 task, probably because it relies on random parameter perturbations (i.e. a GA) for optimization, which does not scale to high-dimensional controllers [47]. ME-ES EXPLORE performs poorly, as it is not interested in performance, but only in exploring the behavioral space, most of which is not useful. The exploitation ability of ME-ES EXPLORE only relies on its directed exploration component to find higher-performing solutions by chance. Indeed, neither ME-GA nor ME-ES EXPLORE leverages directed exploitation. ME-ES EXPLOIT only focuses on exploitation, but still performs undirected exploration by retaining novel solutions that fill new cells. While ME-ES EXPLORE-EXPLOIT targets performance only half of the time, it still finds a *BM* that is good for damage recovery. Its exploration component enables better coverage of the behavioral space while its exploitation component ensures high performing controllers are in the map. Note that a good *BC* space coverage is a poor indicator of adaptation performance, whereas having a high-performing, somewhat populated map is a good indicator. Directed exploitation seems to be required to achieve good performance in the Ant-v2 task. Undirected exploration –as performed by ME-ES EXPLOIT– is sufficient to build a behavioral map useful for damage recovery, as

its recovery performance is similar to that of variants using directed exploration and exploitation (ME-ES EXPLORE-EXPLOIT).

4.2 Deep Exploration

4.2.1 Domains. We use two modified domains from OpenAI Gym based on *Humanoid-v2* and *Ant-v2* (Fig. 4). In *Deceptive Humanoid*, the humanoid robot faces a U-shaped wall (like in [12]), while in *Ant Maze* the Ant is placed in a maze similar to [19]. Both environments possess a strongly deceptive reward, whose gradient leads the agent directly into a trap. For both environments, we use the final (x, y) position of the agent as the BC. In *Deceptive Humanoid*, the fitness is defined as the cumulative reward, a mixture of velocity along the x -axis and costs for joint torques. See Brockman et al. [8] for more details. In *Ant Maze*, the fitness is the Euclidean distance to the goal (green area). Controllers are run for 3000 timesteps in *Ant Maze* and up to 1000 timesteps in *Deceptive Humanoid*, less if the agent falls over.

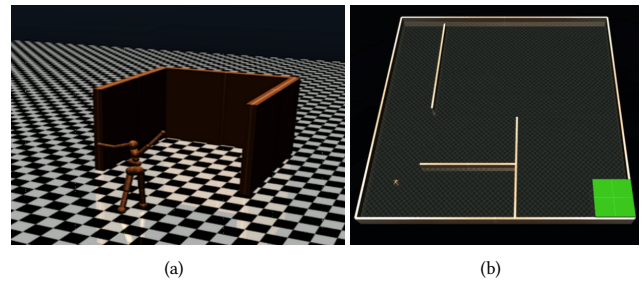


Figure 4: Domains for the deep exploration study. a: Deceptive Humanoid domain. b: Ant Maze domain. Here the goal is the green area and the trap is in the cul-de-sac to the right of the agent’s starting position.

4.2.2 Results - Deceptive Humanoid. Undirected exploration (ME-ES EXPLOIT) is insufficient to get out of the trap and directed exploration alone (ME-ES EXPLORE) falls short as well (Fig. 5a). That said, like NS-ES, ME-ES EXPLORE eventually explores around the wall, as indicated by its greater than 3000 performance. Algorithms

implementing directed exploration and directed exploitation manage to go around the wall and achieve high performance (Fig. 5a), regardless of whether they use decoupled (ME-ES EXPLORE-EXPLOIT) or coupled (NSR-ES, NSRA-ES) exploration-exploitation. Surprisingly, ME-ES EXPLORE displays poor map coverage despite its focus on exploration (Fig. 5b).

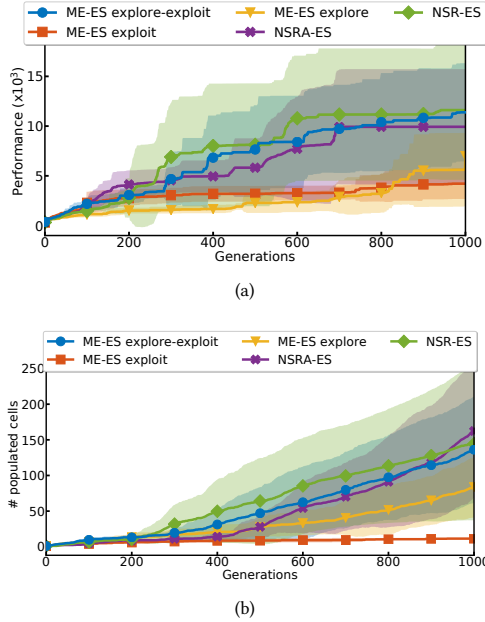


Figure 5: Deceptive Humanoid behavioral map. a: Map performance (best in map). Being stuck in the trap corresponds to around 3000. b: Map coverage (# populated cells).

4.2.3 Results - Ant Maze. In the *Ant Maze*, both ME-ES EXPLOIT and NSR-ES get stuck in the deceptive trap, as indicated by a score of -27 . All other methods (ME-ES EXPLORE, NS-ES, ME-ES EXPLORE-EXPLOIT, and NSRA-ES) are able to avoid the trap and obtain a score closer to 0. Examining the exploitation ratio of NSRA-ES, we observe that all runs quickly move towards performing mostly exploration. That said, some runs have a ratio that falls to pure exploration and stays there, whereas in others the algorithm manages to find a high-performing controller, which triggers an increase in preference towards performance (Fig. 7)

4.2.4 Interpretation.

Deceptive Humanoid. In this environment, simply following the fitness objective can only lead the agent into the U-shaped trap. Because the environment is open-ended, pure exploration does not guarantee the discovery of a high performing behavior either, as it is possible to endlessly explore in directions misaligned with performance, a phenomenon previously encountered with NS-ES in Conti et al. [12]. This explains the poor performance of both ME-ES EXPLORE and ME-ES EXPLOIT. However, combining exploration and exploitation provides the best of both worlds, and we observe that all algorithms that do so navigate around the wall (NSR-ES,

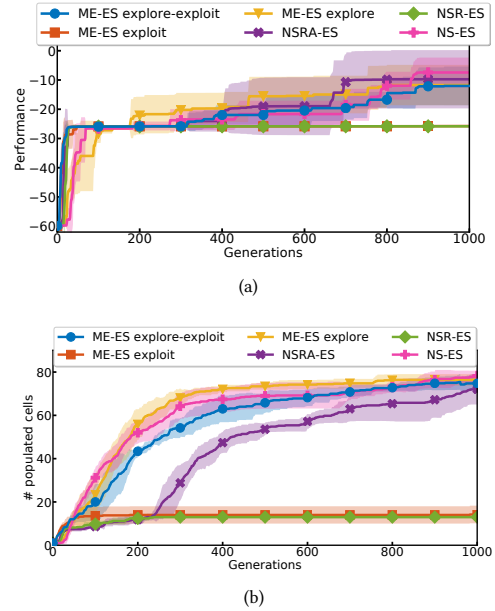


Figure 6: Ant Maze behavioral map. a: Map performance (negative distance to the goal area in map). The trap corresponds to -27 . b: Map coverage (# populated cells).

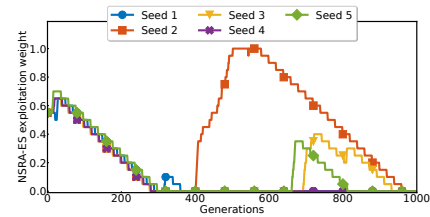


Figure 7: NSRA-ES weight adaptation. Evolution of NSRA-ES weight tuning the ratio between the performance (exploitation) and the novelty (exploration) objectives for 5 runs. Higher values correspond to higher levels of exploitation.

NSRA-ES, ME-ES EXPLORE-EXPLOIT). As there is no need to follow gradients orthogonal to those of the performance objective to succeed, coupling exploration and exploitation (e.g. NSR-ES) is sufficient to achieve high fitness. The poor map coverage of ME-ES EXPLORE may be a result of it only optimizing for performance indirectly via MAP-Elites map updates. As a result, it is never encouraged to learn to walk efficiently, and thus unable to fill large portions of the map. In Conti et al. [12] however, NS-ES manages to make the humanoid walk using pure directed exploration. This may be because NS-ES only updates a population of 5 controllers, whereas any controller in the behavioral map of ME-ES EXPLORE can be updated. A given ME-ES EXPLORE controller will thus be updated less frequently than any of the 5 NS-ES controllers. This dilution of the number of updates of each controller might explain why ME-ES EXPLORE is less efficient at exploring than NS-ES in this environment. Because the environment is unbounded, different

runs may explore different direction of the BC space, resulting in high standard deviations.

Ant Maze. In the Ant Maze task, pure exploitation will lead the agent to the deceptive trap (Fig. 4a). However, the environment is enclosed by walls so the agent cannot explore endlessly. Thus, performing pure exploration will eventually lead to the goal. This explains the success of directed exploration algorithms (NS-ES, ME-ES EXPLORE) and the poor performance of directed exploitation algorithms (ME-ES EXPLOIT). This environment requires a more extensive exploration procedure to achieve the goal than *Deceptive Humanoid*. In *Ant Maze*, the agent needs to avoid two traps and needs to make turns to achieve its goal, while in *Humanoid Deceptive*, the agent can simply walk with an angle of 45 degrees to achieve high scores. As such, an even mix of the performance and novelty objectives (NSR-ES) fails, as we try to average two contradicting forces. NSR-ES has a better chance of getting out of the trap by adding more and more exploration to the mix until the agent escapes. In doing so, however, the mixing weight can go down to $w = 0$ and only consider the exploration objective, thus losing sight of the goal (Fig. 7). ME-ES EXPLORE-EXPLOIT is able to escape the trap because of its decoupled objectives; exploration steps will lead it to explore the environment until it reaches a point where exploitation steps allow it to reach the goal.

5 DISCUSSION AND CONCLUSION

In this paper we present ME-ES, a new QD algorithm scaling the powerful MAP-Elites algorithm to deep neural networks by leveraging ES. We present three variants of this algorithm: (1) ME-ES EXPLOIT, where ES targets a performance objective, (2) ME-ES EXPLORE, where ES targets a novelty objective, and (3) ME-ES EXPLORE-EXPLOIT, where ES targets performance and novelty objectives in an alternating fashion. Because the MAP-Elites behavioral map implicitly implements undirected exploration (update rule 1) and undirected exploitation (update rule 2), these variants implement directed exploitation with undirected exploration, directed exploration with undirected exploitation, and decoupled directed exploration and exploitation, respectively.

In a first application, we use ME-ES to curate an archive of diverse and high-performing controllers that is subsequently used for damage adaptation. We show that ME-ES manages to recover from joint damage in a high-dimensional control task while ME-GA does not. In a second application, we use ME-ES to solve strongly deceptive exploration tasks. Here, we show that ME-ES EXPLORE-EXPLOIT performs well in both open and closed domains, on par with the state-of-the-art exploration algorithm NSR-ES. As in most DRL control tasks, the controllers optimized in this work contain about 10^5 parameters [24, 43]. Previous work showed ES could scale to even larger controllers (e.g. $\approx 10^6$ in Atari games) [43], which suggests ME-ES could as well.

Others have conducted work similar to ours. Frans et al. [19] tackles an environment similar to Ant Maze with hierarchical RL. Their method pre-trains independent low-level controllers to crawl in various directions before learning a high-level control to pick directions to find the goal. On top of requiring some domain knowledge to train low-level controllers, hierarchical RL has trouble composing sub-controllers, a problem that is solved by resetting the agent to a

default position between sub-sequences. In this work, however, we learn to solve the deep exploration problem by directly controlling joints.

Concurrently to our work, Fontaine et al. [17] proposed a similar algorithm called CMA-ME, in which MAP-Elites is combined with CMA-ES, another algorithm from the ES family [25]. In addition to a performance objective, CMA-ME considers two others. The *improvement* objective selects for controllers that discovered new cells or improved over already populated cells. The *random direction* objective rewards controllers for minimizing the distance between their BC and a behavioral target vector randomly sampled from the behavioral space. Although the last objective does encourage exploration, it is not explicitly directed towards novelty as in ME-ES EXPLORE and is not decoupled from performance. Additionally, although CMA-ES is generally recognized as a very powerful algorithm, in practice it is limited to comparatively low-dimensional parameter spaces due to the algorithmic complexity induced by its covariance matrix updates ($O(n^2)$).

Because ME-ES only affects the optimization procedure of MAP-Elites, any application leveraging behavioral maps could benefit from its improved optimization efficiency. In addition to the demonstrated applications for exploration (Section 4.2) or damage adaptation (Section 4.1), previous works have used MAP-Elites for maze navigation [41] or to explore the behavioral space of soft robotic arms and the design space of soft robots [34]. It can also co-evolve a repertoire of diverse images matching each of the ImageNet categories [35] or generate diverse collections of 3D objects [29].

ME-ES could also leverage advances proposed for either MAP-Elites or ES. Vassiliades et al. [49], for example, propose to use a centroidal Voronoi tessellation to scale MAP-Elites to high-dimensional behavioral spaces, while Chatzilygeroudis et al. [10] improve on the behavioral adaptation procedure to enable adaptation without the need to reset the environment (or for humans to intervene in the case of real robots). Finally, Gajewski et al. [21] recently proposed to replace the performance objective of ES by an *evolvability* objective, which aims to maximize the effect of parameter perturbations on behavior. Evolvability can make a good exploration objective because perturbing *evolvable* controllers is likely to increase the speed at which a behavioral map can be explored.

More generally, ME-ES could leverage a set of various objectives in parallel. As the bottleneck lies in the computation of the 10^4 offspring, various objectives can be computed and optimized for simultaneously (novelty, performance, combinations of them as in NSR-ES, evolvability etc.). Each resulting controller can be tested 30 times and considered a candidate for the behavioral map updates, at a negligible increase in cost of $N_{Objectives} \times 30 \ll 10^4$.

In the *deep exploration* experiment, ME-ES could also be extended to perform hierarchical evolution. In such an algorithm, solutions are *chains* of controllers, where each link is a controller run for a given amount of time. After sampling a cell and retrieving the associated chain, the algorithm can either mutate the last controller of the chain or add a new controller to it. This would implement the *go-explore* strategy proposed in Ecoffet et al. [15], where the agent first returns to a behavioral cell before exploring further.

The addition of ME-ES and its variants to the QD family opens new avenues, as the exploration and diversity properties of QD algorithms can now be scaled to high-dimensional problems.

Implementation. A python implementation of ME-ES is available at <https://github.com/uber-research/Map-Elites-Evolutionary>.

REFERENCES

- [1] Joshua Achiam and Shankar Sastry. 2017. Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv preprint arXiv:1703.01732* (2017).
- [2] Thomas Back, Frank Hoffmeister, and Hans-Paul Schwefel. 1991. A survey of evolution strategies. In *Proceedings of the fourth international conference on genetic algorithms*, Vol. 2. Morgan Kaufmann Publishers San Mateo, CA.
- [3] Adrien Baranes and Pierre-Yves Oudeyer. 2013. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems* 61, 1 (2013), 49–73.
- [4] Andrew G Barto. 2013. Intrinsic motivation and reinforcement learning. In *Intrinsically motivated learning in natural and artificial systems*. Springer, 17–47.
- [5] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. 2016. Unifying count-based exploration and intrinsic motivation. In *Advances in neural information processing systems*. 1471–1479.
- [6] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47 (2013), 253–279.
- [7] Lucas Beyer, Damien Vincent, Olivier Teboul, Sylvain Gelly, Matthieu Geist, and Olivier Pietquin. 2019. MULEX: Disentangling Exploitation from Exploration in Deep RL. *arXiv preprint arXiv:1907.00868* (2019).
- [8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).
- [9] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. 2018. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894* (2018).
- [10] Konstantinos Chatzilygeroudis, Vassilis Vassiliadis, and Jean-Baptiste Mouret. 2018. Reset-free trial-and-error learning for robot damage recovery. *Robotics and Autonomous Systems* 100 (2018), 236–250.
- [11] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. 2018. Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms. *arXiv preprint arXiv:1802.05054* (2018).
- [12] 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*. 5027–5038.
- [13] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. 2015. Robots that can adapt like animals. *Nature* 521, 7553 (2015), 503.
- [14] Antoine Cully and Yiannis Demiris. 2017. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation* 22, 2 (2017), 245–259.
- [15] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. 2019. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995* (2019).
- [16] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR.org, 1126–1135.
- [17] Matthew C Fontaine, Julian Togelius, Stefanos Nikolaidis, and Amy K Hoover. 2019. Covariance Matrix Adaptation for the Rapid Illumination of Behavior Space. *arXiv preprint arXiv:1912.02400* (2019).
- [18] Sébastien Forestier, Yoan Mollard, and Pierre-Yves Oudeyer. 2017. Intrinsically motivated goal exploration processes with automatic curriculum learning. *arXiv preprint arXiv:1708.02190* (2017).
- [19] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. 2017. Meta learning shared hierarchies. *arXiv preprint arXiv:1710.09767* (2017).
- [20] Scott Fujimoto, Herke Van Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477* (2018).
- [21] Alexander Gajewski, Jeff Clune, Kenneth O Stanley, and Joel Lehman. 2019. Evolvability ES: scalable and direct optimization of evolvability. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 107–115.
- [22] Adam Gleave, Michael Dennis, Neel Kant, Cody Wild, Sergey Levine, and Stuart Russell. 2019. Adversarial policies: Attacking deep reinforcement learning. *arXiv preprint arXiv:1905.10615* (2019).
- [23] 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*. 249–256.
- [24] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290* (2018).
- [25] Nikolaus Hansen. 2016. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772* (2016).
- [26] Rein Houthoofd, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. 2016. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*. 1109–1117.
- [27] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
- [28] Ayaka Kume, Eiichi Matsumoto, Kuniyuki Takahashi, Wilson Ko, and Jethro Tan. 2017. Map-based multi-policy reinforcement learning: enhancing adaptability of robots by deep reinforcement learning. *arXiv preprint arXiv:1710.06117* (2017).
- [29] Joel Lehman, Sebastian Risi, and Jeff Clune. 2016. Creative generation of 3D objects with deep learning and innovation engines. In *Proceedings of the 7th International Conference on Computational Creativity*.
- [30] Joel Lehman and Kenneth O Stanley. 2008. Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*. 329–336.
- [31] Joel Lehman and Kenneth O Stanley. 2011. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM, 211–218.
- [32] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).
- [33] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).
- [34] Jean-Baptiste Mouret and Jeff Clune. 2015. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909* (2015).
- [35] Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. 2015. Innovation engines: Automated creativity and improved stochastic optimization via deep learning. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 959–966.
- [36] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. 2018. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720* (2018).
- [37] Pierre-Yves Oudeyer, Frédéric Kaplan, and Verena V Hafner. 2007. Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation* 11, 2 (2007), 265–286.
- [38] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. 2017. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 16–17.
- [39] Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. 2019. Teacher algorithms for curriculum learning of Deep RL in continuously parameterized environments. *arXiv preprint arXiv:1910.07224* (2019).
- [40] Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. 2016. Searching for quality diversity when diversity is unaligned with quality. In *International Conference on Parallel Problem Solving from Nature*. Springer, 880–889.
- [41] Justin K Pugh, Lisa B Soros, Paul A Szerlip, and Kenneth O Stanley. 2015. Confronting the challenge of quality diversity. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 967–974.
- [42] Ingo Rechenberg. 1973. Evolutionsstrategie—Optimierung technischer Systeme nach Prinzipien der biologischen Information. *Stuttgart-Bad Cannstatt: Friedrich Frommann Verlag* (1973).
- [43] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864* (2017).
- [44] Jürgen Schmidhuber. 1991. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*. 222–227.
- [45] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. 2010. Parameter-exploring policy gradients. *Neural Networks* 23, 4 (2010), 551–559.
- [46] Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. 2018. Model-based active exploration. *arXiv preprint arXiv:1810.12162* (2018).
- [47] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. 2017. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567* (2017).
- [48] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.
- [49] Vassilis Vassiliadis, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. 2017. Using centroidal voronoi tessellations to scale up the multidimensional archive of phenotypic elites algorithm. *IEEE Transactions on Evolutionary Computation* 22, 4 (2017), 623–630.
- [50] 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)*. IEEE, 3381–3387.
- [51] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.
- [52] Jingwei Zhang, Niklas Wetzl, Nicolai Dorka, Joschka Boedecker, and Wolfram Burgard. 2019. Scheduled intrinsic drive: A hierarchical take on intrinsically motivated exploration. *arXiv preprint arXiv:1903.07400* (2019).