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

Bryon Tjanaka
tjanaka@usc.edu
University of Southern California
Los Angeles, California, USA

Matthew C. Fontaine
mfontain@usc.edu
University of Southern California
Los Angeles, California, USA

David H. Lee
dhlee@usc.edu
University of Southern California
Los Angeles, California, USA

Yulun Zhang
yulunzhang@cmu.edu
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Nivedit Reddy Balam
nbalam@usc.edu
University of Southern California
Los Angeles, California, USA

Nathaniel Dennler
dennler@usc.edu
University of Southern California
Los Angeles, California, USA

Sujay S. Garlanka
garlanka@usc.edu
University of Southern California
Los Angeles, California, USA

Nikitas Dimitri Klapsis
nklapsis@usc.edu
University of Southern California
Los Angeles, California, USA

Stefanos Nikolaidis
stefanosnikolaidis@gmail.com
University of Southern California
Los Angeles, California, USA

ABSTRACT

Recent years have seen a rise in the popularity of quality diversity (QD) optimization, a branch of optimization that seeks to find a collection of diverse, high-performing solutions to a given problem. To grow further, we believe the QD community faces two challenges: developing a framework to represent the field's growing array of algorithms, and implementing that framework in software that supports a range of researchers and practitioners. To address these challenges, we have developed pyribs, a library built on a highly modular conceptual QD framework. By replacing components in the conceptual framework, and hence in pyribs, users can compose algorithms from across the QD literature; equally important, they can identify unexplored algorithm variations. Furthermore, pyribs makes this framework simple, flexible, and accessible, with a user-friendly API supported by extensive documentation and tutorials. This paper overviews the creation of pyribs, focusing on the conceptual framework that it implements and the design principles that have guided the library's development. Pyribs is available at https://pyribs.org

CCS CONCEPTS

• **Computing methodologies → Search methodologies**; • **Software and its engineering → Software libraries and repositories**.

KEYWORDS

quality diversity, framework, software library

ACM Reference Format:
Bryon Tjanaka, Matthew C. Fontaine, David H. Lee, Yulun Zhang, Nivedit Reddy Balam, Nathaniel Dennler, Sujay S. Garlanka, Nikitas Dimitri Klapsis, and Stefanos Nikolaidis. 2023. pyribs: A Bare-Bones Python Library for Quality Diversity Optimization. In *Genetic and Evolutionary Computation Conference (GECCO '23), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3583131.3590374

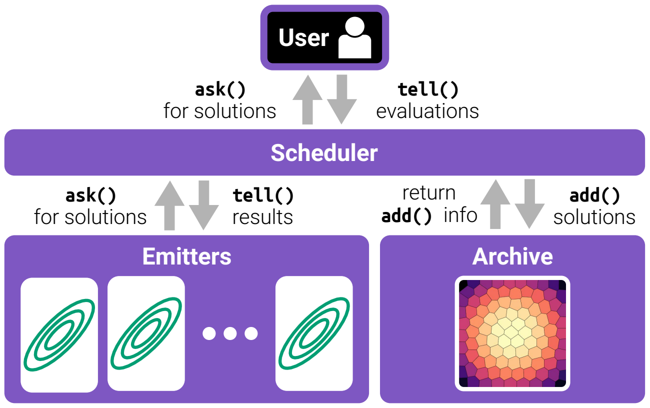

Figure 1: Pyribs implements the RIBS framework for QD optimization. The user first `ask()`'s for solutions from a *scheduler*. The scheduler selects *emitters* to `ask()` for solutions and returns the solutions to the user. After evaluating the solutions, the user `tell()`'s the results to the scheduler. The scheduler `add()`'s the solutions to the *archive* and receives information that it `tell()`'s to the emitters, enabling the emitters to update their internal search state.

1 INTRODUCTION

Many research problems decompose into highly contextual components that prevent one solution from working well across all possible situations. In such cases, developing a set of solutions

Table 1: By selecting different components in the RIBS framework, we can compose a variety of recent algorithms from the QD literature and test them in pyribs. Furthermore, we can identify combinations of components which may lead to new algorithms. Refer to Sec. 3.2 for more details on the archives, emitters, and schedulers shown here.

	Archive				Emitters					Scheduler	
	Grid	CVT	Sliding Boundaries	Unstructured	Gaussian	Iso+LineDD	CMA-ES	Genetic Algorithm	Gradient Arborescence	Basic	Bandit
MAP-Elites [55]	×				×					×	
CVT-MAP-Elites [69]		×			×					×	
Iso+LineDD MAP-Elites [70]		×				×				×	
MESB [27]			×		×					×	
NSLC [50]				×				×		×	
CMA-ME [32]	×						×			×	
CMA-MAE [30]	×						×			×	
ME-MAP-Elites [14]	×					×	×				×
CMA-MEGA [29]	×								×	×	
CMA-MAEGA [30]	×								×	×	

rather than a single solution enables researchers to account for a range of contexts. For instance, a roboticist may develop diverse walking gaits so that their robot can adapt to different morphological considerations [15], while a video game designer may generate multiple video game levels so that players can experience various levels of difficulty [22, 28], and a chemist may create multiple viable drug candidates which exhibit unique properties [71].

Quality diversity (QD) optimization [8] addresses such problems by searching for collections of diverse, high-performing solutions. Originating in neuroevolution with Novelty Search [49, 50] and MAP-Elites [55], QD has grown to become a general-purpose optimization paradigm with applications in a number of areas. As of writing, there are at least 167 papers on the topic [12], spanning areas as diverse as reinforcement learning [10, 11, 59, 62, 67, 68], robot manipulation [53, 54], human-robot interaction [25, 26, 31], video game level generation [22, 28], agent testing [5], generative modeling [29], urban planning [37], design [34], internet congestion control [23], and drug discovery [71]. QD has also moved outside of publications and into more popular forms of media like blog posts [2, 24, 33, 44, 52, 72, 74] and conference tutorials [9, 17–19].

To grow further, we believe the QD community must overcome two challenges. The first challenge is to develop a conceptual framework capable of implementing the wide and growing range of QD algorithms. Many QD algorithms contain interchangeable components, and a unified framework allows for mixing several state-of-the-art components into new algorithms as the field advances. To this end, previous work has proposed the Unifying Modular Framework (UMF) [16] to connect the two main families of QD algorithms, Novelty Search and MAP-Elites. However, UMF was primarily designed for QD algorithms based on genetic operators [20], which limits its applicability to recently developed QD algorithms that have a strong optimization component, such as algorithms which incorporate Evolution Strategies (ES) [10, 11, 32, 67, 68], gradient ascent [29, 30], or Bayesian Optimization [45].

The second challenge is to implement this framework in software which can support a wide range of users, ranging from beginners entering the field to experienced researchers seeking to develop new algorithms. Historically, the conception of flexible, well-documented software libraries has been quintessential to the blooming of popular research areas. For instance, PyTorch [61] and TensorFlow [1] have catalyzed the development and deployment of countless deep learning algorithms in academia and industry [43], and pycma [41] has popularized the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) as one of the standard tools of evolutionary computation. Such libraries have profound effects on their respective fields because not only do they provide powerful features concealed with an expressive, user-friendly application programming interface (API), but they also make these features accessible through comprehensive documentation and tutorials, enabling new practitioners to incorporate the latest algorithms into their projects. Thus far, the QD community has introduced a number of its own libraries. While these libraries have successfully spurred research, they are targeted towards researchers *within* the QD community, offering them high performance [51, 56], reference implementations [57], or a rich end-to-end experience [7].

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

RIBS is highly *modular*: As Table 1 shows, *many existing QD algorithms can be composed by replacing individual components of the framework.* The table also highlights unexplored gaps that could be filled by combining different components, indicating potentially promising areas for future research. Yet, the modular design does not sacrifice simplicity, a key feature in attracting new practitioners.

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

- **Simple:** Centered *only* on components that are absolutely necessary to run a QD algorithm, allowing users to combine the framework with other software frameworks.
- **Flexible:** Capable of representing a wide range of current and future QD algorithms, allowing users to easily create or modify components.

[1]The name "RIBS" stems from the title of Fontaine et al. [32], "Covariance Matrix Adaptation for the **R**apid **I**llumination of **B**ehavior **S**pace," which introduced the concepts of emitters and schedulers. The name "pyribs" is thus a combination of "Python" and "RIBS." The proper spelling of pyribs is all-lowercase, similar to pycma [41], except at the beginning of sentences, when it is capitalized as Pyribs.

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

Pyribs offers modular components that can be assembled into a QD algorithm and controlled with an API inspired by pycma's ask-tell interface [41]. It also features extensive documentation, including tutorials (Fig. 3) demonstrating its usage.[2] Since its inception in 2021, pyribs has grown to support the research of at least a dozen groups across academia and industry worldwide. As of writing, it has been applied to image generation [29, 30], video game level generation [22], environment generation [5], reinforcement learning [67, 68], hyperparameter optimization [65], architecture design [36], and internet congestion control [23].

2 BACKGROUND

2.1 Quality Diversity

2.1.1 Focus. The pyribs library focuses on continuous optimization problems over the search space $\mathbb{R}^n$, the same class of problems targeted by the pycma [41] library. By focusing only on continuous optimization, the library becomes less abstract as search vectors become explicitly defined. Yet, continuous optimization contains an expressive class of problems that the QD community cares about.

2.1.2 Definition. We define the continuous QD problem. We assume an *objective function* $f : \mathbb{R}^n \to \mathbb{R}$ and k *measure functions*[3] $m_i : \mathbb{R}^n \to \mathbb{R}$, represented jointly as $\boldsymbol{m} : \mathbb{R}^n \to \mathbb{R}^k$. We let $S = \boldsymbol{m}(\mathbb{R}^n)$ be the measure space formed by the range of $\boldsymbol{m}$.

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

$$\begin{aligned} \max \quad & f(\boldsymbol{\theta}) \\ \text{subject to} \quad & \boldsymbol{m}(\boldsymbol{\theta}) = \boldsymbol{s} \qquad \forall \boldsymbol{s} \in S \end{aligned} \tag{1}$$

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

A special case of the QD problem is the *differentiable quality diversity (DQD)* [29] problem, where the objective and measure functions are first-order differentiable with gradients $\boldsymbol{\nabla} f$ and $\boldsymbol{\nabla m}$.

2.1.3 Algorithms. We consider two alternatives of what constitutes a *representative* solution in the QD problem definition (Eq. 1), resulting in two families of algorithms.

Algorithms based on MAP-Elites [55] tessellate the measure space S into M cells, and Θ is constrained such that each of its solutions falls into a different cell of the tessellation based on its measure values. The vanilla MAP-Elites [55] mutates randomly sampled solutions in the archive with a genetic operator; generated solutions are added to the archive if their objective value exceeds that of the solution currently occupying their corresponding archive cell. Since its inception, MAP-Elites extensions have included new genetic operators, such as the Iso+LineDD operator inspired by crossover [70], as well as new methods for tessellating the measure space to create the archive. For example, MAP-Elites with Sliding Boundaries (MESB) adapts the size of grid cells online to reflect the distribution of solutions in measure space [27], while CVT-MAP-Elites [69] precomputes a centroidal Voronoi tessellation (CVT) [21] of the measure space that defines the archive cells.

Algorithms based on Novelty Search [49, 50] maintain an unstructured archive where each solution must be *novel* by being a certain distance away from its nearest neighbors in measure space. A genetic algorithm then optimizes a population of solutions to achieve further novelty. While Novelty Search itself is a purely diversity-driven approach, many of its successors are designed for QD; for instance, Novelty Search with Local Competition (NSLC) [50] balances between optimizing for the objective and novelty via multi-objective evolutionary algorithms.

QD algorithms have started to incorporate modern optimization algorithms. For example, Covariance Matrix Adaptation MAP-Elites (CMA-ME) [32] directly optimizes for the QD objective with CMA-ES [40]. In QD optimization, it is efficient to search *multiple* regions of the measure space simultaneously, while balancing the exploration of each region. Therefore, CMA-ME introduced the concepts of *emitters* and *schedulers*. Each emitter maintains a separate CMA-ES instance, while the scheduler balances how emitters explore each measure space region. Emitters and schedulers became core components of the RIBS framework (Sec. 3). Subsequent works building on CMA-ME include Covariance Matrix Adaptation MAP-Annealing (CMA-MAE) [30], which adds an *archive learning rate* to the MAP-Elites grid archive. The learning rate regulates how quickly a non-stationary discount function changes, resulting in a *soft archive* that balances the tradeoff between pure optimization and exploration. In addition, CMA-MEGA and CMA-MAEGA (CMA-ME / CMA-MAE via a Gradient Arborescence) [29, 30] address DQD problems with similar principles as CMA-ME and CMA-MAE.

Finally, Multi-Emitter MAP-Elites (ME-MAP-Elites) [14] introduced a new scheduler by modifying the method for selecting emitters. While the scheduler in CMA-ME maintains several CMA-ES emitters and a round-robin emitter scheduler, ME-MAP-Elites maintains an *emitter pool* consisting of emitters from CMA-ME and emitters that apply the Iso+LineDD operator [70]. Every iteration, the scheduler uses a multi-armed bandit selector from prior work [35] to select emitters which are likely to improve the archive.

2.2 The Unifying Modular Framework

The Unifying Modular Framework (UMF) [16], an early conceptual QD framework, proposed to unite the components of the two pioneering algorithms in QD optimization: MAP-Elites and NSLC.

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

UMF unified under one framework the two major families of QD algorithms: MAP-Elites and NSLC. However, UMF was proposed when all QD algorithms were based on genetic algorithms, and the framework is not expressive enough to represent modern QD

[2]Website: https://pyribs.org
Source Code: https://github.com/icaros-usc/pyribs
Documentation and Tutorials: https://docs.pyribs.org
[3]Prior work refers to measure function outputs as "behavior characteristics," "behavior descriptors," or "feature descriptors." We use the "measures" terminology in pyribs.

algorithms based on other optimization methods. Specifically, UMF incorporates a selector, which chooses solutions as inputs to genetic operators. While selectors can retain a population of solutions, they are not suitable for optimization algorithms that require an internal state, e.g., an evolution path in CMA-ES [40] or momentum in Adam [47]. Drawing from the architecture of CMA-ME [32], our proposed RIBS framework incorporates emitters, which were designed to encapsulate any optimization algorithm used to generate solutions.

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

2.3 Existing QD Libraries

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

2.3.1 Sferes$_{v2}$. Sferes$_{v2}$ [56] is a C++ framework for evolutionary computation that also supports QD algorithms. Sferes$_{v2}$ is primarily designed for high performance, leveraging template-based meta-programming to provide an efficient object-oriented interface and offering multi-core parallel execution through Intel TBB and MPI. While the template-based structure results in significant performance benefits, it limits accessibility for non-expert users. In comparison, pyribs focuses solely on QD algorithms rather than on general evolutionary computation. It is a Python-based library that emphasizes accessibility over performance (Sec. 4.1.3).

2.3.2 QDpy. QDpy [7] is designed to be a feature-rich Python library for QD optimization. Besides supporting ready-to-go implementations of algorithms such as MAP-Elites and CMA-ME, QDpy provides building blocks that can be assembled into new algorithms. To run a QD algorithm, a QDpy user instantiates a *container* (i.e., an archive) and passes it to an *algorithm* object. The user then defines an evaluation function and passes the function to the QDpy system to optimize. QDpy also provides logging and plotting utilities and tools to run the evaluation function on distributed computation.

QDpy's flexibility is limited by the requirement that users pass in an evaluation function. While passing in this function allows users to leverage QDpy's various utilities, this requirement also makes it difficult for users to integrate their own utilities. In contrast, pyribs provides an ask-tell interface where users handle evaluations on their own (Sec. 4.2.4). Essentially, pyribs focuses on components necessary for running QD algorithms, allowing users to integrate tools and frameworks with which they are already familiar.

2.3.3 pymap_elites. pymap_elites [57] provides customizable reference implementations of MAP-Elites and its variants CVT-MAP-Elites [69], MAP-Elites with the Iso+LineDD operator [70], and Multi-task MAP-Elites [58]. Unlike pymap_elites, pyribs offers a larger selection of algorithms under one framework.

2.3.4 QDax. QDax [51] is a recent library that was developed after the initial release of pyribs. The library focuses on efficient QD, reinforcement learning (RL), and evolutionary algorithm implementations for hardware accelerators such as GPUs and TPUs, taking advantage of the parallel nature of these methods. QDax specializes in reinforcement learning and robotics domains, where evaluation remains an expensive bottleneck. Many experiments that took hours or days on a CPU cluster take only minutes with GPU acceleration in QDax. To leverage accelerators in both function evaluation and algorithm implementation, QDax builds on the JAX library [6] and provides a JAX-based API.

While pyribs incorporates batch operations like those found in QDax to ensure a reasonable level of performance (Sec. 4.2.5), pyribs only runs single-threaded on a single CPU (Sec. 4.2.3). In addition, pyribs is not based on specialized libraries, which makes it accessible to a more general audience, such as beginners who have only basic Python knowledge and limited computational resources. Finally, while QDax extends beyond QD by providing baseline algorithms from RL and multi-objective optimization, pyribs focuses on general-purpose QD algorithms under the RIBS framework.

3 THE RIBS FRAMEWORK

Pyribs implements the conceptual RIBS framework that consists of three core components: (1) an *archive* storing solutions generated by the QD algorithm, (2) *emitters* generating solutions for evaluation, and (3) a *scheduler* managing the interaction of the archive and emitters and providing the primary ask-tell [41] interface to the user. Algorithm 1 shows the standard execution loop for combining these components. As we show in Sec. 3.2.2, this execution loop is flexible and not limited to a single call to the ask-tell interface.

3.1 Components

3.1.1 Archive. The archive is a data structure which stores solutions generated by the QD algorithm, along with any information relevant to solutions, such as objective and measure values. The primary archive method is `add()`, which takes in multiple solutions with their objective and measure values, attempts to add them to the collection of solutions, and returns information about the addition. Examples of such information include "status" (whether the solution found a new cell in the archive, improved an existing cell, or was not added at all), "novelty" (the average distance in measure space from the solution to its k-nearest neighbors in the archive [49]), and "improvement value" (the difference between the solution's objective value and that of the solution which it replaced [30]). Archives may support additional functionality, such as methods for sampling solutions and retrieving solutions with given measure values.

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

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

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

One example of a RIBS emitter is the CMA-ES emitter from CMA-ME [32]. Here, calling `ask()` samples solutions from the Gaussian distribution maintained by CMA-ES, while calling `tell()` updates the Gaussian distribution and internal CMA-ES parameters [40].

It is also possible that emitters in RIBS do not require any internal state. For instance, when `ask()` is called, one variation of MAP-Elites generates new solutions by sampling existing archive solutions and perturbing them with fixed-variance Gaussian noise. Since there are no parameters to update for this Gaussian noise mutation, the `tell()` method does not perform any operation.

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

Schedulers implement an ask-tell interface as shown in Algorithm 1. When `ask()` is called (line 11), the scheduler selects one or more emitters and calls each emitter's `ask()` method to generate solutions. When `tell()` is called (line 18), the scheduler takes in the objective and measure function evaluations of these solutions and `add()`'s the solutions to the archive. Then, the scheduler passes the solutions, evaluations, and archive addition information to the emitters via each emitter's `tell()` method.

In the original emitter implementation [32], emitters directly called `add()` to insert solutions into the archive. However, allowing emitters to modify the archive meant that feedback from `add()` depended on the order in which emitters were called, similar to adding solutions sequentially in archives as discussed in Sec. 3.1.1. Now, although the emitters may read data from the archive (e.g., when sampling solutions), only the scheduler calls `add()` and passes the returned information to the emitters through their `tell()` method.

Ultimately, the scheduler provides the primary user interface in the RIBS framework. As shown in Algorithm 1, users directly call `ask()`, evaluate solutions, and pass the results to `tell()`.

3.2 Composing Algorithms in RIBS

Algorithm 1 shows a standard execution loop in RIBS. First, the user configures the core components. Then, in the main loop (line 6), the user calls the scheduler's ask-tell interface and evaluates solutions in between the calls. Importantly, the RIBS components (archive, emitters, and scheduler) in this loop are interchangeable, and the execution loop can be customized to support new QD algorithms.

Algorithm 1: Standard Execution Loop in RIBS

```
1  QD Algorithm ($n_e, n_{it}$):
     Input: Number of emitters $n_e$, number of iterations $n_{it}$,
            parameters for $Archive$, $Emitters$, and $Scheduler$
     Result: Generates solutions to optimize the QD
             objective, stored in an $Archive$
2    $Archive \leftarrow$ init_archive()
3    $[Emitter_1..Emitter_{n_e}] \leftarrow$ init_emitters($Archive$)
4    $Scheduler \leftarrow$ init_scheduler($Archive$,
5                                   $[Emitter_1..Emitter_{n_e}]$)
6    for $itr \leftarrow 1..n_{it}$ do
7        $L \leftarrow Scheduler$.ask()
8        $User$ computes $Evals = [f(\theta), \boldsymbol{m}(\theta)$ for $\theta$ in $L]$
9        $Scheduler$.tell($Evals$)
10   return $Archive$
11 Scheduler.ask ():
     Result: Returns a list of solutions $L$ generated by the
             emitters.
12   $L \leftarrow []$                          // Empty list
13   for $i \leftarrow 1..n_e$ do
14       if $Emitter_i$ should generate solutions then
15           $L_i \leftarrow Emitter_i$.ask()
16           $L \leftarrow LL_i$               // Concatenate $L_i$ to $L$
17   return $L$
18 Scheduler.tell ($Evals$):
     Input: Objective and measure function evaluations of
            the list of solutions $L$.
     Result: Inserts solutions into $Archive$ and updates
             $Emitters$.
19   $add\_info \leftarrow Archive$.add($L, Evals$)
20   for $i \leftarrow 1..n_e$ do
21       if $Emitter_i$ generated solutions then
22           Retrieve solutions $L_i$ generated by $Emitter_i$
23           Retrieve $Evals_i$ corresponding to $L_i$
24           Retrieve $add\_info_i$ corresponding to $L_i$
25           $Emitter_i$.tell($L_i, Evals_i, add\_info_i$)
```

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

3.2.1 Integrating Different Components. First, we consider algorithms which replace components of RIBS without modifying the standard execution loop of Algorithm 1. Table 1 summarizes the components required for each algorithm. Throughout this section, we italicize the components listed in Table 1 as we introduce them.

We begin with MAP-Elites [55], which has a *grid archive* that tessellates the measure space into a grid. MAP-Elites incorporates a single emitter that randomly selects solutions from the archive and applies mutations. One kind of mutation is to add Gaussian noise; in this case, we call the emitter the *Gaussian emitter.* As is common in many versions of MAP-Elites, the Gaussian emitter can also sample directly in the solution space on initial calls to `ask()`. Since this emitter has no adaptive components, its `tell()` method does nothing. Finally, MAP-Elites has a *basic scheduler* that simply selects this emitter on every iteration.

Replacing components creates different MAP-Elites variants. Substituting the Gaussian emitter with the *Iso+LineDD emitter*, which applies the Iso+LineDD operation [70], results in Iso+LineDD MAP-Elites.[4] We can replace the archive with a *CVT archive* or *sliding boundaries archive* to obtain CVT-MAP-Elites [69] and MESB [27].

We can also consider methods based on Novelty Search like NSLC [50]. Here, the *unstructured archive* adds solutions if they are far away from their k nearest neighbors in the archive. Meanwhile, the *genetic algorithm emitter* contains a genetic algorithm such as NEAT [66] that manages a population of solutions. In contrast to the Gaussian and Iso+LineDD emitters, the genetic algorithm emitter's `tell()` method updates its internal population. The scheduler remains the same as in MAP-Elites.

CMA-ME [32] and CMA-MAE [30] are more complicated because they require managing multiple instances of CMA-ES in parallel. In this case, we create multiple *CMA-ES emitters*, each with their own CMA-ES instance. Calling `ask()` on each emitter samples solutions from CMA-ES's multivariate Gaussian distribution, and calling `tell()` updates the distribution parameters and the internal CMA-ES parameters. We combine these emitters with the grid archive and basic scheduler from MAP-Elites.

Multi-Emitter MAP-Elites (ME-MAP-Elites) [14] provides an example of an algorithm that requires a different scheduler. The default ME-MAP-Elites includes CMA-ES and Iso+LineDD emitters. Its *bandit scheduler* applies a multi-armed bandit algorithm to select a subset of these emitters based on whether they have previously generated solutions that were inserted into the archive.

3.2.2 Modifying the execution loop. Besides algorithms that replace components of RIBS, we also consider those that modify the RIBS execution loop. In this regard, CMA-MEGA [29] and CMA-MAEGA [30] both require a *gradient arborescence emitter*, which constructs solutions by branching from a solution point based on the objective and measure gradients. This branching requires calling `ask()` and `tell()` twice: once to collect the solution point and return its evaluations and gradients, and once to handle the branched solutions. Compared to Algorithm 1, we add another set of calls to `ask()` and `tell()` in the loop on line 6, with appropriate arguments to handle passing gradients back to `tell()`.

In addition, a number of recent works [5, 34, 45, 73] integrate surrogate models with QD algorithms in domains where evaluations are expensive. Surrogate-assisted QD algorithms construct an archive based on evaluations predicted by a surrogate model and then select candidate solutions for ground-truth evaluations.

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

[4]The original Iso+LineDD MAP-Elites [70] uses a CVT archive, but the authors noted that a grid archive would also work with their algorithm.

Algorithm 2: QD Algorithm with Surrogate Model in RIBS

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

4 DESIGNING PYRIBS

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

4.1 Principles

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

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

4.1.3 Accessible. Pyribs is accessible to a wide audience, ranging from beginners to experienced researchers, by having readable source code, being easy to install, and having full documentation defining its usage. Our dependency choices ensure that beginners

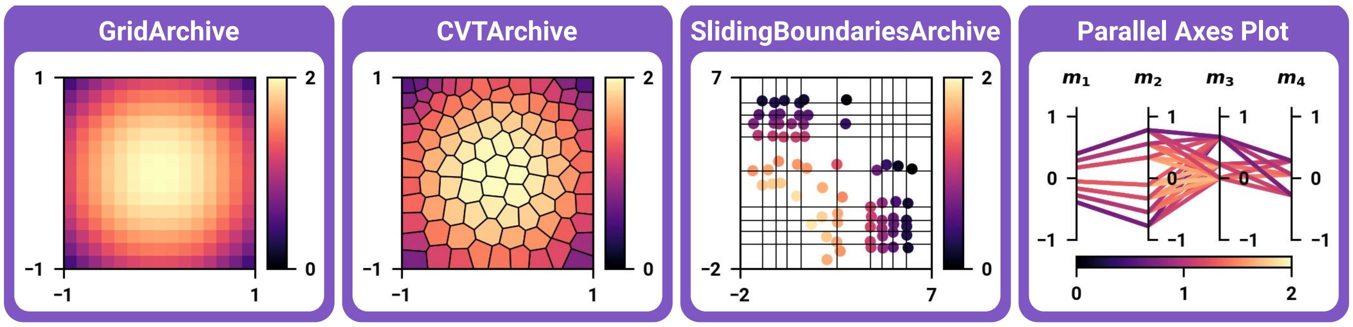

Figure 2: Pyribs visualization tools. We show example 2D heatmaps, where the axes correspond to the measure values, and the color of each archive cell indicates its objective value. In `SlidingBoundariesArchive`, the points show the locations of solutions in measure space, and the lines show the grid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

with limited computational resources or basic hardware can install pyribs and study the tutorials. The library also supports experienced researchers by being amenable to modifications.

4.2 Implementation Features

These features demonstrate how our implementation choices align with the design principles of section 4.1.

4.2.1 Choice of Python. Python offers many desirable features. Beyond being a beginner-friendly language, it has a flourishing ecosystem, with package repositories like the Python Package Index (PyPI) [64] and Anaconda [3] providing easy access to many useful libraries. While Python itself is slower than lower-level languages like C++, libraries like NumPy [42] compensate for this limitation by providing access to efficient numerical computation routines. Python can also integrate with other programming languages through various packages; for instance, PyJNIus [48] enables running Python-based QD algorithms [5, 28] with the Mario AI Framework [46], a benchmark implemented in Java. Furthermore, Python has become the *de facto* language of machine learning, and with the influx of QD applications to machine learning [28, 29, 59, 73], it is important to support QD researchers from that area. Thus, implementing the RIBS framework in Python and distributing pyribs on PyPI and Anaconda makes pyribs *accessible*, as users can easily install and learn to use the library.

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

There are many other solution encodings that a QD library could support. For instance, discrete solutions (e.g., a list of integers) can be implemented with the same arrays used in pyribs, but recent research in QD has focused on continuous domains [8]. Alternatively, a QD library could support objects such as solutions of variable length, graphs [71], or neural network structure encodings from NEAT [66]. Although the RIBS framework is not limited to any one solution encoding, such structures are often domain-specific, and including them would increase the complexity of pyribs.

4.2.3 Single CPU. To be *simple* and *accessible*, pyribs runs single-threaded on a single CPU. Thus, pyribs can run on hardware ranging from laptops to high-performance clusters. While being single-threaded may limit the performance of pyribs, the runtime in many QD problems is dominated by the user's evaluation of solutions, rather than by the execution of the QD algorithm in pyribs. Moreover, the need for high-performance algorithm implementations is already fulfilled by libraries like Sferes$_{v2}$ [56] and QDax [51]. However, if internal algorithm runtime grows to be a bottleneck in QD problems, we could redesign pyribs to optionally parallelize execution by putting each emitter on a separate thread. Note that while pyribs itself runs on a single CPU, evaluations are left to the user and may be parallelized as described in the next section.

4.2.4 Evaluations are left to the user. Since evaluations are often the bottleneck in QD, we considered providing utilities for running evaluations of solutions in parallel, as is done in QDpy [7]. We decided against doing so since many evaluation functions require specific dependencies and hardware configurations that are difficult to support in a general-purpose library. In short, we maintain *simplicity* by leaving evaluations to the user. Nevertheless, our documentation includes basic examples of how to integrate parallelism into pyribs workflows. For instance, our tutorial "Using CMA-ME to Land a Lunar Lander Like a Space Shuttle" parallelizes evaluations in only two lines of code with Python's `multiprocessing` module.

4.2.5 Batch operations. The initial version of pyribs implemented many operations sequentially, including the archives' `add()` methods (Sec. 3.1.1). To compensate for the reduced performance, we added the just-in-time compiler Numba [4] to many of these functions. However, multiple users indicated that doing so decreased *accessibility* by making the library difficult to modify and debug. Hence, in the most recent version of pyribs (0.5.0), we have re-implemented these methods as batch operations without Numba. For instance, instead of adding one solution to the archive at a time, pyribs now leverages NumPy array operations to add a batch of solutions simultaneously. These operations improve code readability

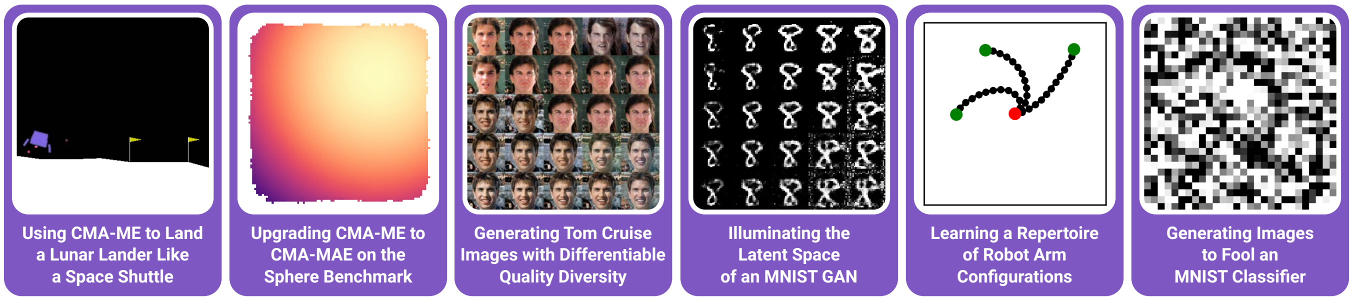
Using CMA-ME to Land a Lunar Lander Like a Space Shuttle
Upgrading CMA-ME to CMA-MAE on the Sphere Benchmark
Generating Tom Cruise Images with Differentiable Quality Diversity
Illuminating the Latent Space of an MNIST GAN
Learning a Repertoire of Robot Arm Configurations
Generating Images to Fool an MNIST Classifier

Figure 3: Tutorials enable pyribs users to quickly learn about the library and experiment with problems from the QD literature.

and performance over the sequential implementations while still operating on a single CPU. For instance, on the 20D sphere linear projection benchmark, the runtime of CMA-ME decreased from 140s with sequential+Numba to 60s with batch implementations.

4.2.6 One-layer hierarchy. Ideally, every pyribs component would be implemented in a single file with no dependencies. Doing so would make the source code highly *accessible*, as a user could easily read the code for a component and modify it, similar to pymap_elites [57]. However, since components often share functionality, standalone files would lead to duplicate code and hamper maintenance. We compromise by introducing a one-layer hierarchy, where archives, emitters, and schedulers all inherit from their respective base classes. For instance, a user can learn about the implementation of `GridArchive` by reading the source code for `ArchiveBase` and `GridArchive`. This hierarchy helps make pyribs *flexible*, as users who create new components can inherit from these base classes instead of re-implementing basic functionality.

4.2.7 Visualization tools. Since there are no commonly available visualization tools for archives, we added our own to make pyribs *accessible*. As shown in Fig. 2, pyribs features heatmap visualizations for all its archives, as well as a parallel axes plot method that can visualize an archive of any dimensionality.

4.2.8 Documentation and tutorials. A key feature for increasing *accessibility* in pyribs is its extensive documentation and tutorials. Every pyribs component is documented in detail, and pyribs has an array of tutorials (Fig. 3). These tutorials teach users about the library and introduce them to common QD problems, such as latent space illumination [28], the arm repertoire benchmark [16, 70], and the sphere linear projection benchmark [32].

4.2.9 Industry standard practices. We draw from industry standard style guides [38] to promote readability and correctness in our source code. We automatically format our code with yapf [39] and check for basic errors with Pylint [63]. Furthermore, we implement a comprehensive suite of unit tests. Since it is difficult to test stochastic components like emitters and schedulers, our total code coverage by unit tests[5] as of version 0.5.0 is 81%, but on archives, which are nearly deterministic, our coverage is 97%. Finally, when implementing new components, we run them on benchmarks such as sphere linear projection [32] to verify that our implementation matches results from prior work. These practices ensure that future changes in pyribs do not affect existing functionality.

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

5 CONCLUSION

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

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

On the other end of the spectrum, we aim to serve the needs of more experienced researchers by further developing the capabilities of pyribs. A major avenue in this direction is to expand the collection of pyribs components. Pyribs currently centers on the MAP-Elites family, but potential additions outside this family include the unstructured archive from Novelty Search, the archive with learned measures from AURORA [13], and emitters and schedulers from NS-ES [11] and SERENE [60].

Simultaneously, it is important for pyribs to support integrations with other fields. For example, many recent QD papers combine QD with deep learning [5, 28–30, 59]. Given the prevalence of GPUs in deep learning, it could thus be helpful to add GPU support to overcome delays incurred by transferring data between CPU and GPU. However, such advanced features will need to be delicately balanced against our design principles.

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

ACKNOWLEDGMENTS

This work was partially supported by the NSF CAREER (#2145077), NSF NRI (#2024949), and NSF GRFP (#DGE-1842487). We thank the anonymous reviewers, Varun Bhatt, Ya-Chuan Hsu, and Robby Costales for their invaluable feedback.

REFERENCES

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (USA, 2016), OSDI'16, USENIX Association, p. 265–283.

[2] Allard, M. Quality-diversity algorithms. https://maximeallard.lu/2021/03/24/quality-diversity-algorithms/, 03 2021. Retrieved 2023-01-31.

[3] Anaconda, Inc. Anaconda. https://anaconda.org.

[4] Anaconda Inc. Numba: A just-in-time compiler for numerical functions in python. https://numba.pydata.org. Computer software.

[5] Bhatt, V., Tjanaka, B., Fontaine, M., and Nikolaidis, S. Deep surrogate assisted generation of environments. In *Advances in Neural Information Processing Systems* (2022), S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, Curran Associates, Inc., pp. 37762–37777.

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

[7] Cazenille, L. Qdpy: A python framework for quality-diversity. https://gitlab.com/leo.cazenille/qdpy, 2018.

[8] Chatzilygeroudis, K., Cully, A., Vassiliades, V., and Mouret, J.-B. *Quality-Diversity Optimization: A Novel Branch of Stochastic Optimization.* Springer International Publishing, Cham, 2021, pp. 109–135.

[9] Clune, J., Lehman, J., and Stanley, K. O. Recent advances in population-based search for deep neural networks. ICML 2019 Tutorials, https://youtu.be/g6HiuEnbwJE, 2019.

[10] Colas, C., Madhavan, V., Huizinga, J., and Clune, J. Scaling map-elites to deep neuroevolution. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2020), GECCO '20, Association for Computing Machinery, p. 67–75.

[11] Conti, E., Madhavan, V., Petroski Such, F., Lehman, J., Stanley, K., and Clune, J. 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., 2018, pp. 5027–5038.

[12] Cully, A. Quality-diversity optimisation algorithms. https://quality-diversity.github.io. Retrieved 2023-01-31.

[13] Cully, A. Autonomous skill discovery with quality-diversity and unsupervised descriptors. In *Proceedings of the Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2019), GECCO '19, Association for Computing Machinery, p. 81–89.

[14] Cully, A. Multi-emitter MAP-elites. In *Proceedings of the Genetic and Evolutionary Computation Conference* (jun 2021), ACM.

[15] Cully, A., Clune, J., Tarapore, D., and Mouret, J.-B. Robots that can adapt like animals. *Nature 521* (05 2015), 503–507.

[16] Cully, A., and Demiris, Y. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation 22*, 2 (2018), 245–259.

[17] Cully, A., Mouret, J.-B., and Doncieux, S. Quality-diversity optimisation. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion* (New York, NY, USA, 2020), GECCO '20, Association for Computing Machinery, p. 701–723.

[18] Cully, A., Mouret, J.-B., and Doncieux, S. Quality-diversity optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (New York, NY, USA, 2021), GECCO '21, Association for Computing Machinery, p. 715–739.

[19] Cully, A., Mouret, J.-B., and Doncieux, S. Quality-diversity optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (New York, NY, USA, 2022), GECCO '22, Association for Computing Machinery, p. 864–889.

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

[21] Du, Q., Faber, V., and Gunzburger, M. Centroidal voronoi tessellations: Applications and algorithms. *SIAM Review 41*, 4 (1999), 637–676.

[22] Earle, S., Snider, J., Fontaine, M. C., Nikolaidis, S., and Togelius, J. Illuminating diverse neural cellular automata for level generation. In *Proceedings of the Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2022), GECCO '22, Association for Computing Machinery, p. 68–76.

[23] Endo, T., Abe, H., and Oka, M. Toward automatic generation of diverse congestion control algorithms through co-evolution with simulation environments. In *ALIFE 2022: The 2022 Conference on Artificial Life* (July 2022).

[24] Flageat, M., and Lim, B. Benchmarking quality-diversity algorithms on neuroevolution for reinforcement learning. https://aihub.org/2022/12/14/benchmarking-quality-diversity-algorithms-on-neuroevolution-for-reinforcement-learning/, 12 2022. Retrieved 2023-01-31.

[25] Fontaine, M., Hsu, Y.-C., Zhang, Y., Tjanaka, B., and Nikolaidis, S. On the Importance of Environments in Human-Robot Coordination. In *Proceedings of Robotics: Science and Systems* (Virtual, July 2021).

[26] Fontaine, M., and Nikolaidis, S. A quality diversity approach to automatically generating human-robot interaction scenarios in shared autonomy. *Robotics: Science and Systems* (2021).

[27] Fontaine, M. C., Lee, S., Soros, L. B., De Mesentier Silva, F., Togelius, J., and Hoover, A. K. Mapping hearthstone deck spaces through map-elites with sliding boundaries. In *Proceedings of the Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2019), GECCO '19, Association for Computing Machinery, p. 161–169.

[28] Fontaine, M. C., Liu, R., Khalifa, A., Modi, J., Togelius, J., Hoover, A. K., and Nikolaidis, S. Illuminating mario scenes in the latent space of a generative adversarial network. *Proceedings of the AAAI Conference on Artificial Intelligence 35*, 7 (May 2021), 5922–5930.

[29] Fontaine, M. C., and Nikolaidis, S. Differentiable quality diversity. *Advances in Neural Information Processing Systems 34* (2021).

[30] Fontaine, M. C., and Nikolaidis, S. Covariance matrix adaptation map-annealing. *arXiv preprint arXiv:2205.10752* (2022).

[31] Fontaine, M. C., and Nikolaidis, S. Evaluating human–robot interaction algorithms in shared autonomy via quality diversity scenario generation. *J. Hum.-Robot Interact. 11*, 3 (Sep 2022).

[32] Fontaine, M. C., Togelius, J., Nikolaidis, S., and Hoover, A. K. Covariance matrix adaptation for the rapid illumination of behavior space. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2020), GECCO '20, Association for Computing Machinery, p. 94–102.

[33] Frans, K. Quality diversity: Evolving ocean creatures. https://kvfrans.com/quality-diversity-evolving-ocean-creatures/, 12 2020. Retrieved 2023-01-31.

[34] Gaier, A., Asteroth, A., and Mouret, J.-B. Data-Efficient Design Exploration through Surrogate-Assisted Illumination. *Evolutionary Computation 26*, 3 (09 2018), 381–410.

[35] Gaier, A., Asteroth, A., and Mouret, J.-B. Discovering representations for black-box optimization. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2020), GECCO '20, Association for Computing Machinery, p. 103–111.

[36] Gaier, A., Stoddart, J., Villaggi, L., and Bentley, P. J. T-domino. In *Parallel Problem Solving from Nature – PPSN XVII* (Cham, 2022), G. Rudolph, A. V. Kononova, H. Aguirre, P. Kerschke, G. Ochoa, and T. Tušar, Eds., Springer International Publishing, pp. 263–277.

[37] Galanos, T., Liapis, A., Yannakakis, G. N., and Koenig, R. Arch-elites: Quality-diversity for urban design. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (New York, NY, USA, 2021), GECCO '21, Association for Computing Machinery, p. 313–314.

[38] Google. Google python style guide. https://google.github.io/styleguide/pyguide.html. Retrieved 2023-02-09.

[39] Google. Yapf: A formatter for python files. https://github.com/google/yapf. Computer software.

[40] Hansen, N. The CMA evolution strategy: A tutorial. *CoRR abs/1604.00772* (2016).

[41] Hansen, N., Akimoto, Y., and Baudis, P. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634, Feb. 2019.

[42] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature 585*, 7825 (Sept. 2020), 357–362.

[43] He, H. The state of machine learning frameworks in 2019. https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/, 2019.

[44] Institute of Digital Games. Game ai - creative artificial evolution through quality diversity algorithms. https://www.game.edu.mt/blog/game-ai-creative-artificial-evolution-through-quality-diversity-algorithms/, 04 2019. Retrieved 2023-01-31.

[45] Kent, P., and Branke, J. Bop-elites, a bayesian optimisation algorithm for quality-diversity search. *arXiv preprint arXiv:2005.04320* (2020).

[46] Khalifa, A. Mario AI framework. https://github.com/amidos2006/Mario-AI-Framework, 2019.

[47] Kingma, D. P., and Ba, J. 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* (2015), Y. Bengio and Y. LeCun, Eds.

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

[49] Lehman, J., and Stanley, K. O. Abandoning Objectives: Evolution Through the Search for Novelty Alone. *Evolutionary Computation 19*, 2 (06 2011), 189–223.

[50] Lehman, J., and Stanley, K. O. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (New York, NY, USA, 2011), GECCO

'11, Association for Computing Machinery, p. 211–218.

[51] Lim, B., Allard, M., Grillotti, L., and Cully, A. Accelerated quality-diversity for robotics through massive parallelism. *arXiv preprint arXiv:2202.01258* (2022).

[52] Mohamed, O. Quality-diversity algorithms: Map-polar. https://towardsdatascience.com/quality-diversity-algorithms-a-new-approach-based-on-map-elites-applied-to-robot-navigation-f51380deec5d, 03 2021. Retrieved 2023-01-31.

[53] Morel, A., Kunimoto, Y., Coninx, A., and Doncieux, S. Automatic acquisition of a repertoire of diverse grasping trajectories through behavior shaping and novelty search. In *2022 International Conference on Robotics and Automation (ICRA)* (2022), pp. 755–761.

[54] Morrison, D., Corke, P., and Leitner, J. Egad! an evolved grasping analysis dataset for diversity and reproducibility in robotic manipulation. *IEEE Robotics and Automation Letters 5*, 3 (2020), 4368–4375.

[55] Mouret, J.-B., and Clune, J. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909* (2015).

[56] Mouret, J.-B., and Doncieux, S. SFERESv2: Evolvin' in the multi-core world. In *Proc. of Congress on Evolutionary Computation (CEC)* (2010), pp. 4079–4086.

[57] Mouret, J.-B., et al. Python3 map-elites. https://github.com/resibots/pymap_elites, 2019.

[58] Mouret, J.-B., and Maguire, G. Quality diversity for multi-task optimization. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* (Jun 2020), ACM.

[59] Nilsson, O., and Cully, A. Policy gradient assisted map-elites. In *Proceedings of the Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2021), GECCO '21, Association for Computing Machinery, p. 866–875.

[60] Paolo, G., Coninx, A., Doncieux, S., and Laflaquière, A. Sparse reward exploration via novelty search and emitters. In *Proceedings of the Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2021), GECCO '21, Association for Computing Machinery, p. 154–162.

[61] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems* (2019), H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc.

[62] Pierrot, T., Macé, V., Chalumeau, F., Flajolet, A., Cideron, G., Beguir, K., Cully, A., Sigaud, O., and Perrin-Gilbert, N. Diversity policy gradient for sample efficient quality-diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2022), GECCO '22, Association for Computing Machinery, p. 1075–1083.

[63] Python Code Quality Authority. pylint. http://pylint.pycqa.org/. Computer software.

[64] Python Software Foundation. Python Package Index. https://pypi.org.

[65] Schneider, L., Pfisterer, F., Thomas, J., and Bischl, B. A collection of quality diversity optimization problems derived from hyperparameter optimization of machine learning models. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (New York, NY, USA, 2022), GECCO '22, Association for Computing Machinery, p. 2136–2142.

[66] Stanley, K. O., and Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evolutionary Computation 10*, 2 (2002), 99–127.

[67] Tjanaka, B., Fontaine, M. C., Kalkar, A., and Nikolaidis, S. Training diverse high-dimensional controllers by scaling covariance matrix adaptation map-annealing. *arXiv preprint arXiv:2210.02622* (2022).

[68] Tjanaka, B., Fontaine, M. C., Togelius, J., and Nikolaidis, S. Approximating gradients for differentiable quality diversity in reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2022), GECCO '22, Association for Computing Machinery, p. 1102–1111.

[69] Vassiliades, V., Chatzilygeroudis, K., and Mouret, J.-B. Using centroidal voronoi tessellations to scale up the multidimensional archive of phenotypic elites algorithm. *IEEE Transactions on Evolutionary Computation 22*, 4 (2018), 623–630.

[70] Vassiliades, V., and Mouret, J.-B. Discovering the elite hypervolume by leveraging interspecies correlation. In *Proceedings of the Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2018), GECCO '18, Association for Computing Machinery, p. 149–156.

[71] Verhellen, J., and Van den Abeele, J. Illuminating elite patches of chemical space. *Chem. Sci. 11* (2020), 11485–11491.

[72] Wolz, D. fcmaes - a python-3 derivative-free optimization library. Available at https://github.com/dietmarwo/fast-cma-es, 2022. Python/C++ source code, with description and examples.

[73] Zhang, Y., Fontaine, M. C., Hoover, A. K., and Nikolaidis, S. Deep surrogate assisted map-elites for automated hearthstone deckbuilding. In *Proceedings of the Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2022), GECCO '22, Association for Computing Machinery, p. 158–167.

[74] Zhao, S. Cabbagecat's blogs. https://szhaovas.github.io. Retrieved 2023-01-31.