Journal of Machine Learning Research 25 (2024) 1-16 Submitted 8/23; Revised 1/24; Published 3/24

QDax: A Library for Quality-Diversity and Population-based Algorithms with Hardware Acceleration

Felix Chalumeau* [1] F.CHALUMEAU@INSTADEEP.COM
Bryan Lim* [2] BRYAN.LIM16@IMPERIAL.AC.UK
Raphaël Boige R.BOIGE@INSTADEEP.COM
Maxime Allard M.ALLARD20@IMPERIAL.AC.UK
Luca Grillotti LUCA.GRILLOTTI16@IMPERIAL.AC.UK
Manon Flageat MANON.FLAGEAT18@IMPERIAL.AC.UK
Valentin Macé V.MACE@INSTADEEP.COM
Guillaume Richard G.RICHARD@INSTADEEP.COM
Arthur Flajolet A.FLAJOLET@INSTADEEP.COM
Thomas Pierrot** [1] T.PIERROT@INSTADEEP.COM
Antoine Cully** [2] A.CULLY@IMPERIAL.AC.UK

[1] *InstaDeep* [2] *Department of Computing, Imperial College London*
* *Equal Contribution* ** *Equal Supervision*

Editor: Sebastian Schelter

Abstract

QDax is an open-source library with a streamlined and modular API for Quality-Diversity (QD) optimisation algorithms in JAX. The library serves as a versatile tool for optimisation purposes, ranging from black-box optimisation to continuous control. QDax offers implementations of popular QD, Neuroevolution, and Reinforcement Learning (RL) algorithms, supported by various examples. All the implementations can be just-in-time compiled with JAX, facilitating efficient execution across multiple accelerators, including GPUs and TPUs. These implementations effectively demonstrate the framework's flexibility and user-friendliness, easing experimentation for research purposes. Furthermore, the library is thoroughly documented and has 93% test coverage.

Keywords: Quality Diversity, Population-Based Learning, Evolutionary Computation, Open-Source, JAX, Python

1. Introduction

Quality Diversity (QD) has emerged as a rapidly expanding family of stochastic optimisation algorithms that have demonstrated competitive performance across diverse applications, including robotics (Cully et al., 2015), engineering design (Gaier et al., 2017), and video games (Alvarez et al., 2019). Unlike traditional optimisation algorithms that seek a single objective-maximizing solution, QD algorithms aim to identify a diverse set of such solutions. Recently, the integration of modern deep reinforcement learning techniques with QD (Nilsson and Cully, 2021; Pierrot et al., 2022a) has enabled the tackling of complex problems, such as high-dimensional control and decision-making tasks. This convergence has drawn parallels with other fields that maintain populations of policies and explore diverse behaviors (Eysen-

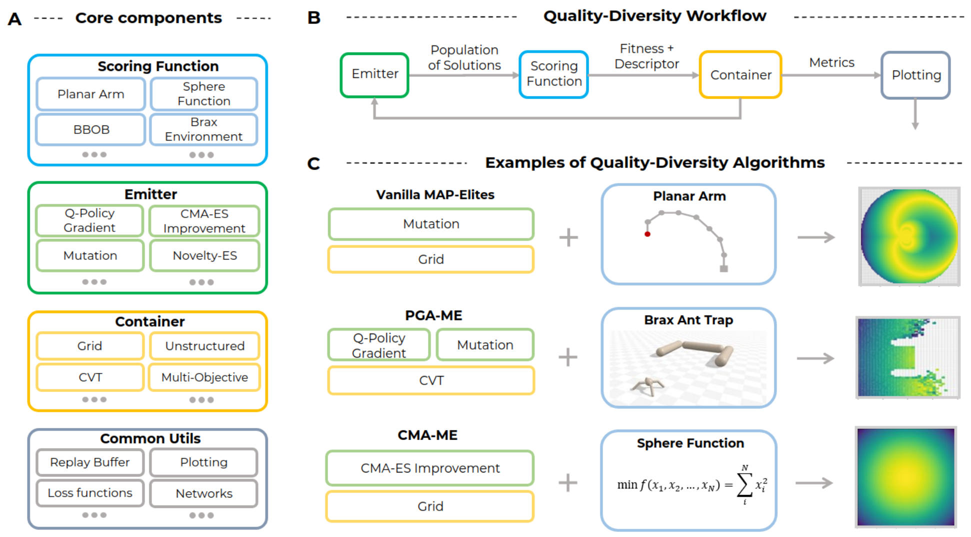


Figure 1: **A**: Core components, used as building blocks to create optimisation experiments. **B**: High-level software architecture of QDax. **C**: Various examples of QD algorithms used for a variety of tasks and problem settings available in QDax.

bach et al., 2018; Sharma et al., 2020). Overall, these algorithmic developments have gained popularity due to their simplicity, versatility, and ability to generate practical solutions for various industrial applications.

To facilitate the unification of algorithms from these distinct research areas, we propose a comprehensive framework embodied in a new open-source library, QDax. QDax encompasses meticulously crafted implementations of 16 methods, prioritising both speed and flexibility. In particular, QDax offers an extensive range of QD algorithm implementations, all within a cohesive framework. Furthermore, QDax provides Reinforcement Learning (RL) algorithms and skill-discovery RL algorithms, both built on the same foundational components. This commonality ensures reliability in comparative analysis and facilitates seamless integration of diverse approaches. Additionally, QDax offers a collection of utilities tailored for benchmarking algorithms on standard tasks, including mathematical functions, robot control scenarios, and industry-driven decision-making problems.

QDax is built on top of JAX (Bradbury et al., 2018), a Python library for high-performance numerical computing and machine learning, to fully exploit modern hardware accelerators like GPUs and TPUs with minimal engineering overhead. JAX enables to harness the scalability advantages of QD methods, as demonstrated in recent studies (Lim et al., 2022), and the parallelisation capabilities of fast parallel simulators, such as those developed in BRAX (Freeman et al., 2021a) and Isaac gym (Makoviychuk et al., 2021). Because it provides reliable implementations and faster benchmarking methods, QDax represents a significant stride towards accelerating the development of QD and population-based methods.

2. QDax Features

Comprehensive baselines and SOTA methods QDax includes numerous baselines for QD, Skill-Discovery RL, Population-Based Training, including state-of-the-art methods from those fields. The performance of implementations is validated through replications of results from the literature. Methods can be run in only a few lines of codes thanks to a simple API. Additionally, the flexibility of this API enables the extension of those methods for research purposes. See Appendix A for examples of API usage.

Fast runtimes. Algorithms and tasks are implemented in JAX and compatible with just-in-time (jit) compilation to take advantage of XLA optimisation and to be able to seamlessly leverage hardware accelerators (e.g. GPUs and TPUs). We compare our implementations with existing ones or reported results. Our main observations are threefold: (i) on similar hardware, we provide significant speed-ups (e.g. QD-PG, factor 5) (ii) ours enable practitioners to run with same time performance algorithms that could only be run with large clusters (e.g ME-ES, 1 GPU vs 1000 CPUs) (iii) one can leverage modern accelerators, resulting in speed-ups of two orders of magnitude (e.g. MAP-Elites, A100). Refer to Appendix D for details.

Optimisation problems. To enhance reproducibility, QDax offers JAX implementations of widely studied problems. QDax includes a collection of fundamental mathematical functions that serves as elementary benchmarks in QD, such as rastrigin, sphere, planar arm, and various others. To further expand the repertoire of tasks, QDax provides a range of utilities for deriving tasks from robotics simulation environments implemented in BRAX. These utilities incorporate robotics motion QD tasks, as inspired by Flageat et al. (2022), and hard exploration tasks, as presented in Chalumeau et al. (2022b). Additionally, we provide support for the RL industrial environments suite Jumanji, (Bonnet et al., 2024), facilitating the evaluation on Combinatorial Optimisation problems. For a comprehensive analysis of the performance of various QDax algorithms on selected tasks, refer to Appendix C.

Documentation, Examples and Code. QDax provides extensive documentation of the entire library.[1] All classes and functions are typed and described with docstrings. Most functions are covered by unit tests (93% code coverage), with publicly available reports.[2] Stability of the library is also ensured through continuous integration, which validates any change through type checking, style checking, unit tests and documentation building. QDax also includes tutorial-style interactive Colab notebooks[3] which demonstrate example usage of the library through the browser without any prior setup. These examples help users get started with the library and also showcase advanced usage. Docker and Singularity container functionalities are also provided to ease reproducibility and deployment on cloud servers. Conda support is also provided and can be used as an alternative. Finally, QDax can be installed via PyPI.[4]

3. Architecture and Design of QDax

QDax is built on Python and JAX (Bradbury et al., 2018). It can seamlessly run on CPUs, GPUs or TPUs on a single machine or in a distributed setting. QDax introduces a framework (see Figure 1) that unifies all the recent state-of-the-art QD algorithms. In this framework, a

1. `https://qdax.readthedocs.io/en/latest/`
2. `https://app.codecov.io/gh/adaptive-intelligent-robotics/QDax`
3. `https://github.com/adaptive-intelligent-robotics/QDax/tree/main/examples`
4. `https://pypi.org/project/qdax/`

Algorithm	**Container**	**Emitter**	**Common utilities**
MAP-Elites	Grid	GA	Network
CMA-ME	Grid	CMA-ES Impr.	Network, CMAES opt.
ME-ES	Grid	ES	Network
PGA-MAP-Elites	CVT	GA, Q-PG	Network, Buffer, Loss
QD-PG	CVT	GA, Q-PG, D-PG	Network, Buffer, Loss
OMG-MEGA	Grid	OMG-MEGA	N/A
CMA-MEGA	Grid	CMA-MEGA	CMAES opt.
MOME	Multi Objective Grid	GA	Pareto front
NSGA-II, SPEA2	Pareto Grid	GA	Pareto front
DIAYN, DADS, SMERL, PBT	N/A	N/A	Network, Buffer, Loss
ME-PBT	Grid	GA, PBT	Network, Buffer, Loss

Table 1: **Comprehensive baselines** implemented in QDax, including state-of-the-art methods for QD, Skill-Discovery, and Multi-Objective Optimisation.

QD algorithm is defined by a **container** and an **emitter**. The **container** defines the way the population is stored and updated at each evolution step. The **emitter** implements the way solutions from the population are updated; this encompasses mutation-based updates, sampling from a distribution (Evolution Strategies), policy-gradient updates, and more. Neuroevolution methods – and more generally algorithms not rooted in QD – also fit in the framework. See Table 1 that highlights components that are shared across algorithms implemented in QDax.

QDax was also designed to compare and combine QD algorithms with other algorithms, such as deep RL and other population-based approaches. Consequently, the architecture ensures common use of many components, such as loss functions, networks and replay buffers. This helps building fair performance comparisons between several classes of algorithms and reduces the time it takes to implement hybrid approaches. The **scoring function** abstraction gives the user flexibility to define the optimisation problem. It can range from standard optimisation functions to complex rollouts in a simulated environment for RL.

4. Comparison to existing libraries

Multiple independent open-source packages are currently accessible for quality diversity (QD). One such framework is Sferes (Mouret and Doncieux, 2010), which offers a selection of QD algorithms implemented in the C++ language. However, it lacks support and integration for deep learning frameworks. On the other hand, python libraries such as Pyribs (Tjanaka et al., 2021) and QDpy (Cazenille, 2018) provide a more extensive range of QD algorithms compared to Sferes. While these libraries are user-friendly, they lack GPU or TPU acceleration and distribution capabilities. Furthermore, they are limited to standard optimisation problems and do not support sequential decision-making problems. Additionally, recent developments in the field of evolutionary algorithms have resulted in the creation of several packages. For instance, EvoJAX (Tang et al., 2022) and evosax (Lange, 2022) are two JAX packages that offer efficient implementations of Evolution Strategies and Neuroevolution methods, as well as the implementation of optimisation problems using JAX. These libraries do not focus on QD and hence have a very limited number of QD methods. Interestingly, these libraries can be used in conjunction with QDax. Similarly, EvoTorch builds upon PyTorch (Paszke et al., 2019) rather than JAX to accelerate Evolutionary Algorithms.

Overall, QDax presents a unified framework in JAX, facilitating rapid development and effortless benchmarking. It serves as a platform for evolutionary, population-based, and diversity-seeking algorithms, while leveraging modern hardware accelerators.

Acknowledgments

We would like to thank Rémi Debette, Achraf Garai, Bryon Tjanaka, Maxence Faldor, and David Braun for their help in this project and all open-source contributors that helped through discussions and feedback. This research was supported with Cloud TPUs from Google's TPU Research Cloud (TRC).

References

Alberto Alvarez, Steve Dahlskog, Jose Font, and Julian Togelius. Empowering quality diversity in dungeon design with interactive constrained map-elites. In *2019 IEEE Conference on Games (CoG)*, page 1–8. IEEE Press, 2019. doi: 10.1109/CIG.2019.8848022. URL `https://doi.org/10.1109/CIG.2019.8848022`.

Clément Bonnet, Daniel Luo, Donal John Byrne, Shikha Surana, Paul Duckworth, Vincent Coyette, Laurence Illing Midgley, Sasha Abramowitz, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries Petrus Smit, Nathan Grinsztajn, Raphael Boige, Cemlyn Neil Waters, Mohamed Ali Ali Mimouni, Ulrich Armel Mbou Sob, Ruan John de Kock, Siddarth Singh, Daniel Furelos-Blanco, Victor Le, Arnu Pretorius, and Alexandre Laterre. Jumanji: a diverse suite of scalable reinforcement learning environments in JAX. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=C4CxQmp9wc`.

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

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

Felix Chalumeau, Raphael Boige, Bryan Lim, Valentin Macé, Maxime Allard, Arthur Flajolet, Antoine Cully, and Thomas PIERROT. Neuroevolution is a competitive alternative to reinforcement learning for skill discovery. In *The Eleventh International Conference on Learning Representations*, 2022a.

Felix Chalumeau, Thomas Pierrot, Valentin Macé, Arthur Flajolet, Karim Beguir, Antoine Cully, and Nicolas Perrin-Gilbert. Assessing quality-diversity neuro-evolution algorithms performance in hard exploration problems. 2022b. doi: 10.48550/ARXIV.2211.13742. URL `https://arxiv.org/abs/2211.13742`.

Cédric Colas, Vashisht Madhavan, Joost Huizinga, and Jeff Clune. Scaling map-elites to deep neuroevolution. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 67–75, 2020.

Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. doi: 10.1109/4235.996017.

DeepMind, Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. URL `http://github.com/google-deepmind`.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Manon Flageat, Bryan Lim, Luca Grillotti, Maxime Allard, Simón C. Smith, and Antoine Cully. Benchmarking quality-diversity algorithms on neuroevolution for reinforcement learning, 2022. URL `https://arxiv.org/abs/2211.02193`.

Manon Flageat, Felix Chalumeau, and Antoine Cully. Empirical analysis of pga-map-elites for neuroevolution in uncertain domains. *ACM Transactions on Evolutionary Learning*, 3 (1):1–32, 2023.

Matthew Fontaine and Stefanos Nikolaidis. Differentiable quality diversity. *Advances in Neural Information Processing Systems*, 34, 2021.

Matthew C Fontaine, Julian Togelius, Stefanos Nikolaidis, and Amy K Hoover. Covariance matrix adaptation for the rapid illumination of behavior space. In *Proceedings of the 2020 genetic and evolutionary computation conference*, pages 94–102, 2020.

C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation, 2021a. URL `http://github.com/google/brax`.

C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax – a differentiable physics engine for large scale rigid body simulation, 2021b. URL `https://arxiv.org/abs/2106.13281`.

Adam Gaier, Alexander Asteroth, and Jean-Baptiste Mouret. Data-efficient exploration, optimization, and modeling of diverse designs through surrogate-assisted illumination.

In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 99–106, 2017.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2023. URL `http://github.com/google/flax`.

Tom Hennigan, Trevor Cai, Tamara Norman, Lena Martens, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020. URL `http://github.com/deepmind/dm-haiku`.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks, 2017. URL `https://arxiv.org/abs/1711.09846`.

Saurabh Kumar, Aviral Kumar, Sergey Levine, and Chelsea Finn. One solution is not all you need: Few-shot extrapolation via structured maxent rl. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 8198–8210. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/5d151d1059a6281335a10732fc49620e-Paper.pdf`.

Robert Tjarko Lange. evosax: Jax-based evolution strategies, 2022. URL `http://github.com/RobertTLange/evosax`.

Bryan Lim, Maxime Allard, Luca Grillotti, and Antoine Cully. Accelerated quality-diversity through massive parallelism. *Transactions on Machine Learning Research*, 2022.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

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

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

Olle Nilsson and Antoine Cully. Policy gradient assisted map-elites. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 866–875, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/`

9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Thomas Pierrot and Arthur Flajolet. Evolving populations of diverse RL agents with MAP-elites. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=CBfYffLqWqb.

Thomas Pierrot, Valentin Macé, Felix Chalumeau, Arthur Flajolet, Geoffrey Cideron, Karim Beguir, Antoine Cully, Olivier Sigaud, and Nicolas Perrin-Gilbert. Diversity policy gradient for sample efficient quality-diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, jul 2022a. doi: 10.1145/3512290.3528845. URL https://doi.org/10.1145%2F3512290.3528845.

Thomas Pierrot, Guillaume Richard, Karim Beguir, and Antoine Cully. Multi-objective quality diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 139–147, 2022b.

Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJgLZR4KvH.

Yujin Tang, Yingtao Tian, and David Ha. Evojax: Hardware-accelerated neuroevolution. *arXiv preprint arXiv:2202.05008*, 2022.

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

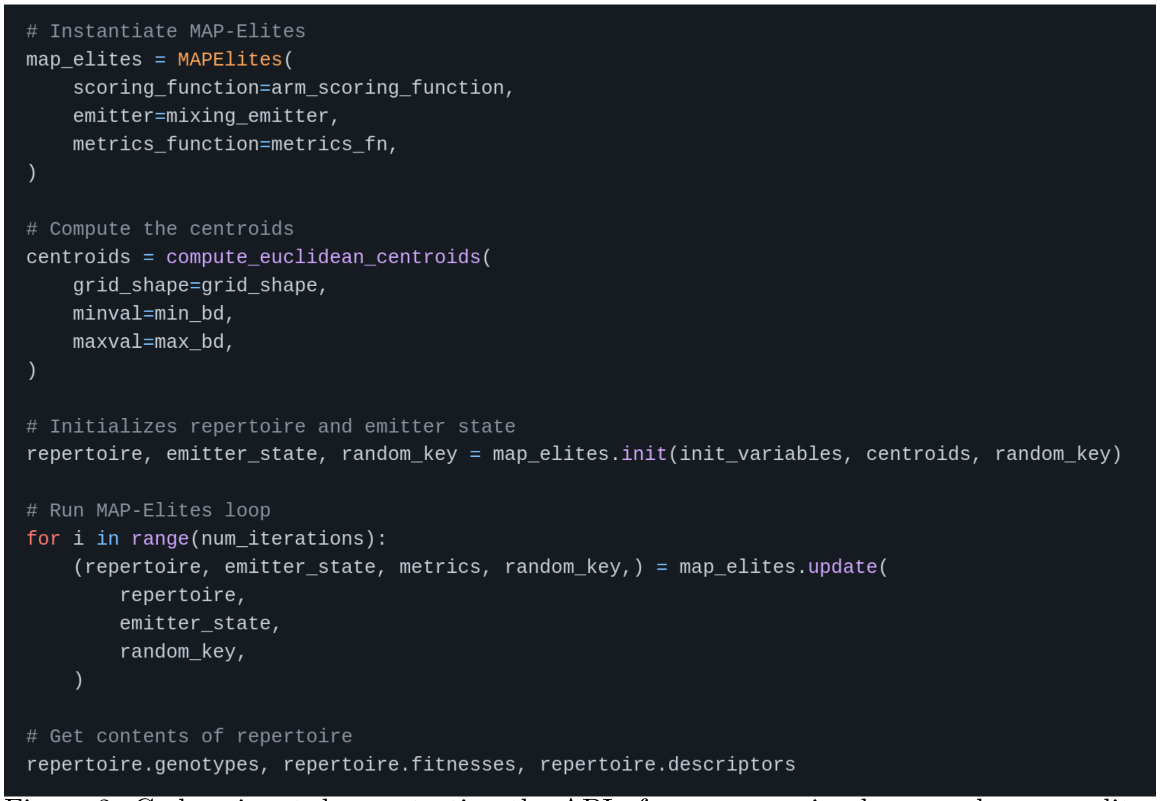
Vassilis Vassiliades, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. Using centroidal voronoi tessellations to scale up the multidimensional archive of phenotypic elites algorithm. *IEEE Transactions on Evolutionary Computation*, 22(4):623–630, 2017.

Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. 2001.

Appendix A. Code API and Usage

Figure 2 shows an example of the code usage demonstrating the simplicity and modularity of the API. This code snippet shows the main steps to run MAP-Elites with QDax, those steps being (i) instantiating an object from the class MAP-Elites (ii) computing the centroids that will be used to define the grid that is to store the solutions produced by the algorithm (iii) initializing the algorithm, which will initialize the repertoire as well as the state of the emitter (iv) running iterations of the update function of MAP-Elites, natively implemented in the class. Interestingly, all it takes to run PGA-MAP-Elites or CMA-ME is to change the type of emitter defined when instantiating the MAP-Elites object on the first line of the code snippet.

For additional illustrations of the API, we strongly encourage the reader to try the examples proposed in QDax: we wrote example notebooks[5] for most algorithms implemented. This should help any user to get started with the library and should also illustrate how to use the library to extend the implemented algorithms for research purpose.



```python
# Instantiate MAP-Elites
map_elites = MAPElites(
    scoring_function=arm_scoring_function,
    emitter=mixing_emitter,
    metrics_function=metrics_fn,
)

# Compute the centroids
centroids = compute_euclidean_centroids(
    grid_shape=grid_shape,
    minval=min_bd,
    maxval=max_bd,
)

# Initializes repertoire and emitter state
repertoire, emitter_state, random_key = map_elites.init(init_variables, centroids, random_key)

# Run MAP-Elites loop
for i in range(num_iterations):
    (repertoire, emitter_state, metrics, random_key,) = map_elites.update(
        repertoire,
        emitter_state,
        random_key,
    )

# Get contents of repertoire
repertoire.genotypes, repertoire.fitnesses, repertoire.descriptors
```

Figure 2: Code snippet demonstrating the API of QDax on a simple example: MAP-Elites used to solve the arm task. User can get their first experiment running with only a few lines of code. Additionally, it only takes a few updates to change the type of algorithm running on the same task.

5. `https://github.com/adaptive-intelligent-robotics/QDax/tree/main/examples`

Algorithm	**Categories**	**References**
MAP-Elites	QD	Mouret and Clune (2015)
CVT MAP-Elites	QD	Vassiliades et al. (2017)
CMA-ME	QD, Evolution Strategies	Fontaine et al. (2020)
ME-ES	QD, Evolution Strategies	Colas et al. (2020)
PGA-MAP-Elites	QD, RL	Nilsson and Cully (2021)
QD-PG	QD, RL	Pierrot et al. (2022a)
DIAYN	Skill Discovery, RL	Eysenbach et al. (2018)
DADS	Skill Discovery, RL	Sharma et al. (2020)
SMERL	Skill Discovery, RL	Kumar et al. (2020)
OMG-MEGA	Differentiable QD	Fontaine and Nikolaidis (2021)
CMA-MEGA	Differentiable QD	Fontaine and Nikolaidis (2021)
MOME	Multi Objective, QD	Pierrot et al. (2022b)
NSGA-II	Multi Objective	Deb et al. (2002)
SPEA2	Multi Objective	Zitzler et al. (2001)
PBT	Population Based, RL	Jaderberg et al. (2017)
ME-PBT	Population Based, QD, RL	Pierrot and Flajolet (2023)

Table 2: Main algorithms implemented in QDax. This table also reports the general category of algorithms each method belongs to, as well as the paper that introduced the algorithm.

Appendix B. Implemented algorithms

The main algorithms implemented in QDax are summarized in table 2. This table provides the name of the algorithm, the broader categories of algorithms that the algorithm belongs to and the reference of the algorithm.

QDax contains state-of-the-art QD algorithms and numerous methods related to Neuroevolution, Skill Discovery, Population-Based learning and Multi-Objective optimisation. In particular, QDax provides the implementation of MAP-Elites (Mouret and Clune, 2015), CVT MAP-Elites (Vassiliades et al., 2017) - corresponding to general QD methods -, CMA-ME (Fontaine et al., 2020), ME-ES (Colas et al., 2020) - described as QD with Evolution Strategies - , PGA-MAP-Elites (Nilsson and Cully, 2021; Flageat et al., 2023), QD-PG (Pierrot et al., 2022a) - which are QD methods with policy gradients, often referred to as QD-RL- , DIAYN (Eysenbach et al., 2018), DADS (Sharma et al., 2020), SMERL (Kumar et al., 2020) - popular methods in RL for Skill Discovery -, OMG-MEGA, CMA-MEGA (Fontaine and Nikolaidis, 2021) - Differentiable QD- , MOME (Pierrot et al., 2022b), NSGA-II, SPEA2- the reference approaches for Multi-Objective optimisation -. All those methods have similar API and can be evaluated on popular tasks. Furthermore, they all take advantage from the speed-up enabled by just-in-time compilation in Jax, making them extremely fast and scalable.

Environment	MAP-Elites	PGA-MAP-Elites	SMERL	DADS	DIAYN
Ant Omni	1.7×10^9	1.85×10^8	1.01×10^7	8.9×10^6	1.01×10^7
Ant Uni	2.45×10^9	1.9×10^8	1.0×10^7	7.8×10^6	9.97×10^6
Ant Maze	1.05×10^9	1.7×10^8	7.09×10^6	8.62×10^6	9.70×10^6
Ant Trap	1.08×10^9	1.3×10^8	9.81×10^6	8.46×10^6	9.73×10^6
Halfcheetah Uni	9.83×10^8	1.39×10^8	1.08×10^7	8.43×10^6	1.07×10^7
Point Maze	2.41×10^9	3.01×10^8	1.09×10^7	9.70×10^6	1.07×10^7
Walker2d Uni	1.95×10^9	2.40×10^8	1.11×10^7	8.94×10^6	1.11×10^7
Average	1.66×10^9	1.93×10^8	9.97×10^6	8.70×10^6	1.03×10^7

Table 3: Number of training steps carried out during two hours of training by the various methods under study on seven BRAX tasks of QDax. Averaged over 5 seeds.

Appendix C. Benchmark Results

This section provides numerous results from the algorithms present in QDax over several benchmarks tasks available in the library. These results can give the reader an idea of the metrics and time performance expected when using QDax. Note that metrics performance are validated against the one reported along original implementations of the algorithms. The reported experiments are taken from Chalumeau et al. (2022a) and were run with a single Quadro RTX 4000 GPU.

The results presented relate to seven tasks from the BRAXRL tasks implemented in QDax. These tasks are benchmark QD-RL tasks used to assess Neuroevolution algorithms (Chalumeau et al., 2022b; Flageat et al., 2022). We refer to them as ANT-UNI, HALFCHEETAH-UNI, WALKER-UNI, ANT-TRAP, ANT-MAZE and POINT-MAZE. Those can be visualized on Figure 3. For more details about the hyper-parameters used, please refer to (Chalumeau et al., 2022a).

For a representative comparison between QD algorithms and Deep RL methods such as SMERL, which do not actively have a population of policies or an archive, a passive archive is used to compute metrics like QD score in those results. Given that there is only a single latent conditioned policy, during training SMERL policy is evaluated by sampling latent codes and recording their trajectories. Behavior descriptors can be extracted from these trajectories, which can then be used for addition to the passive archive. This allows to use similar metrics such as the coverage and QD score when comparing QD and Skill Discovery RL methods.

To demonstrate the speed of our implementations, Table 3 reports the number of training steps achieved in two hours by the implementation of QDax on a single GPU. Figure 5 reports the evolution of fitness, QD score and coverage along time of the algorithms SMERL, MAP-Elites and PGA-MAP-Elites on the seven tasks mentioned above. We also report the evolution of those metrics along environments interactions on Figure 5. Finally, Figure 6 shows the archives of resulting policies from the different algorithms where each cell corresponds to a policy and different behaviors.

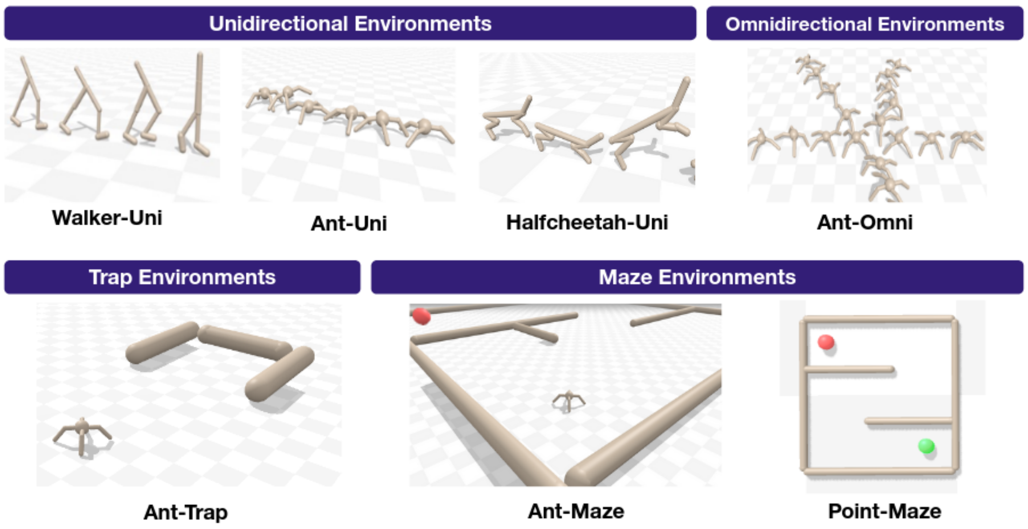


Figure 3: Visualisation of the environments used to show the performances of our implementations of SMERL, PGA-MAP-Elites and MAP-Elites. Diagram is adapted from Chalumeau et al. (2022a); Flageat et al. (2022); Chalumeau et al. (2022b)

Appendix D. Fast implementations

QDax is a package written in JAX, hence all the implementations can be just-in-time compiled (jit) and run on hardware accelerators like GPUs and TPUs with no code overhead. This combines simplicity and efficiency, making QDax particularly suitable for practitioners that do not have access to large compute resources. In this section, we put ourselves in the situation of a practitioner that has a machine with an affordable GPU and wants to use existing available open-source implementations by just installing and running them.

We consider a few open-source implementations of libraries and algorithms which runtime-performance is reported in a paper, reported by its authors, or by simply running the implementation to compare the runtime-performance to the one that someone can expect by running QDax on an affordable GPU (Quadro RTX 4000).

For PGA-MAP-Elites, we compare QDax implementation with the only other available open-source implementation which is the provided author implementation[6]. The author implementation utilizes PyTorch as its deep learning framework. It also uses multiprocessing over CPU devices to handle its environment evaluation and is not directly compatible with GPUs even for training the networks via gradient descent. Hence, to exploit this implementation effectively, a large number of CPUs is required, which is not an option for most practitioners. The original implementation requires approximately 36 hours to perform a run of 10^9 steps with 36 CPUs, which corresponds to 2.8×10^7 steps an hour. Our implementation of PGA-MAP-Elites achieves 10^8 steps an hour. Hence, a practitioner can perform runs of PGA-MAP-Elites in a few hours on his computer instead of several days.

For QD-PG, we compare the runtime performance of QDax's implementation with results reported by the authors of the algorithm. The author implementation is not open-source but they report that with 1 GPU and a dozen CPUs, their implementation can do 10^8 steps

6. `https://github.com/ollenilsson19/PGA-MAP-Elites`

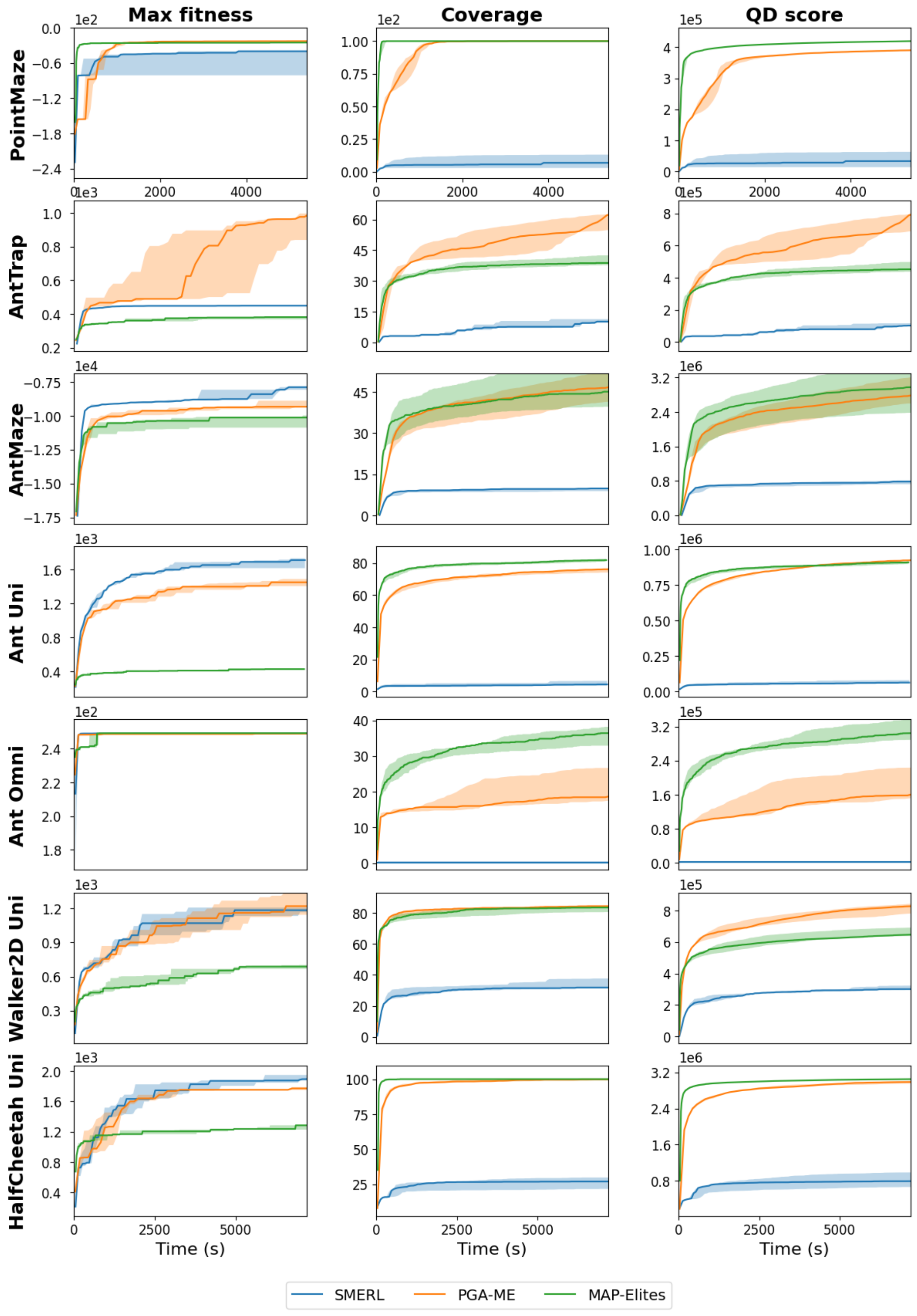


Figure 4: Evolution over wall-clock time of maximum fitness, coverage and QD score during a training phase. Reports algorithms SMERL, PGA-MAP-Elites and MAP-Elites on 2 hours of training.

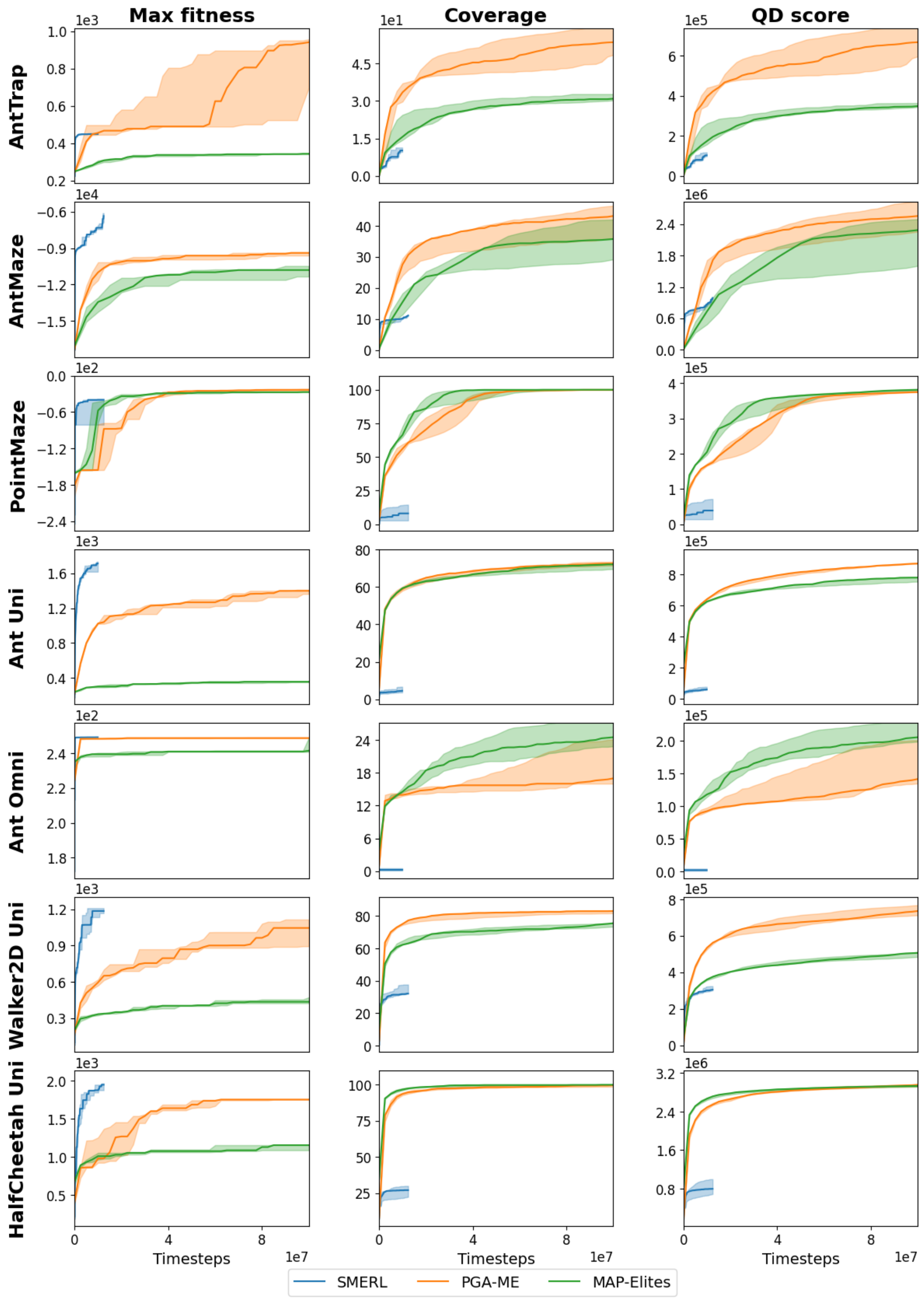


Figure 5: Evolution over environment interactions of maximum fitness, coverage and QD score during a training phase. Reports SMERL on 10^7 timesteps and PGA-MAP-Elites and MAP-Elites on 10^8 timesteps.

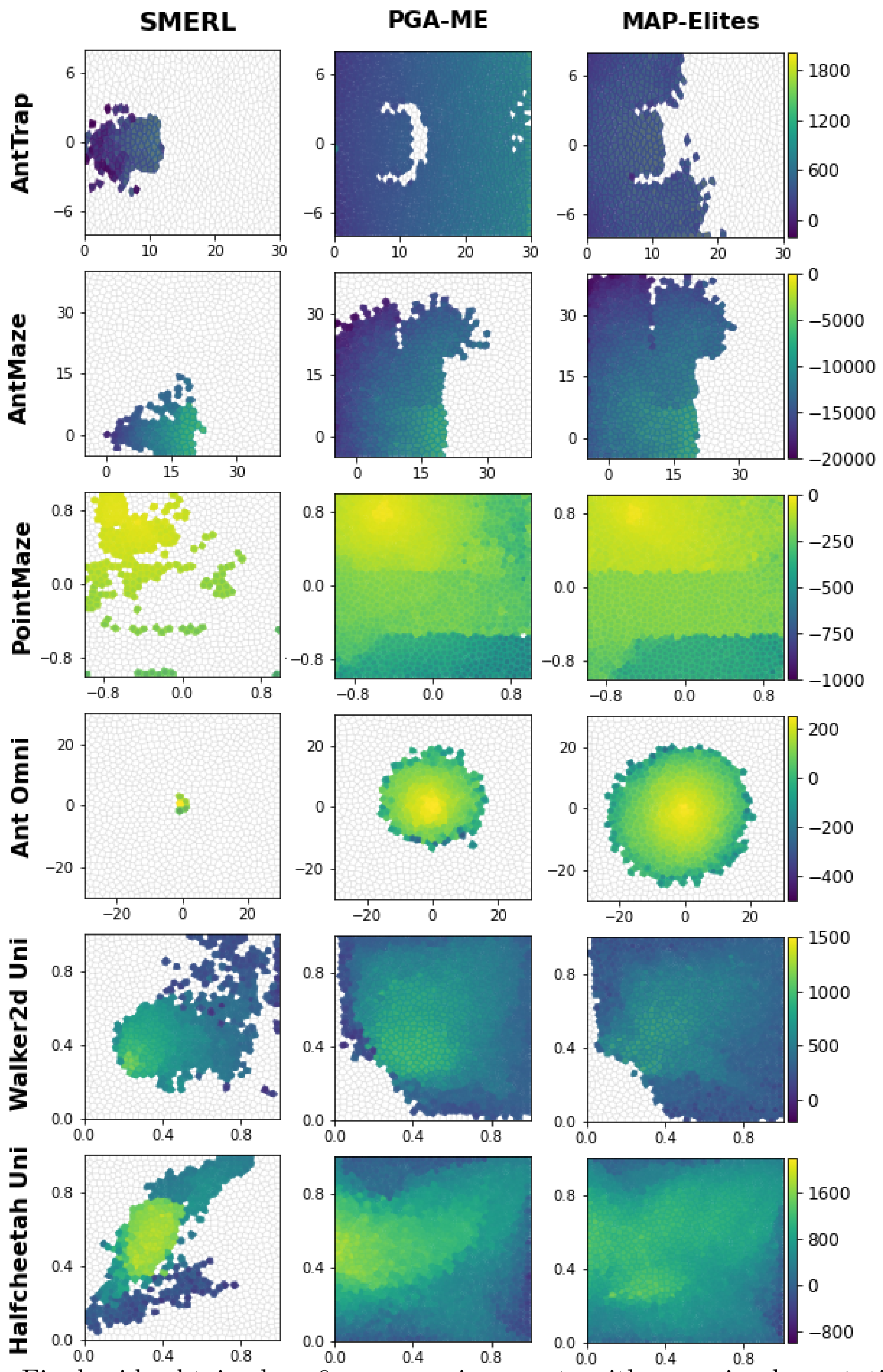


Figure 6: Final grids obtained on 6 BRAX environments with QDax implementations. For ANT-TRAP, ANT-MAZE, POINT-MAZE and ANT-OMNI, the behavior descriptor is the final x, y position. For WALKER-UNI and HALFCHEETAH-UNI, it is the proportion of contact time of the feet on the ground. Reports SMERL, PGA-MAP-Elites and MAP-Elites after two hours of training.

in 15 hours on ANT-TRAP. In QDax, QD-PG can do the same on 3 hours, hence a speed-up of a factor 5.

For ME-ES, we compare our implementation with the original implementation[7]. This implementation requires large computing clusters to be efficient: results reported in the paper (Colas et al., 2020) use 1000 CPUs, which makes it completely unusable for most practitioners. Our implementation can be run on a simple GPU with a few CPUs and still match the runtime performance. The original implementation was reported to perform 6×10^8 environment steps an hour (3×10^{10} in two days for the ANT-MAZE experiment). Our implementation performs 4×10^8 steps an hour.

For MAP-Elites, one can expect to reach up a 100 times speed up compared to alternative open-source implementations. The reader should refer to Lim et al. (2022) for a thorough analysis of the performance to be expected.

Appendix E. Package dependencies

QDax uses several popular packages from the community. In particular:

- BRAX (Freeman et al., 2021b) for fast and jittable physical simulations in Jax, enabling to use continuous control environments.
- CHEX (DeepMind et al., 2020) for type checking.
- FLAX (Heek et al., 2023) as our main library for neural networks.
- DM-HAIKU (Hennigan et al., 2020) to define neural networks in some examples. Hence both FLAX and HAIKU users can found examples in QDax.
- JAX (Bradbury et al., 2018) as the core dependency of the project, enabling high-performance numerical computing.
- JUMANJI (Bonnet et al., 2024) provides RL environments for games on combinatorial optimisation, written in Jax. We provide examples of how to use Jumanji with QDax algorithms.
- OPTAX (DeepMind et al., 2020) for gradient descent algorithms and related utils.

7. `https://github.com/uber-research/Map-Elites-Evolutionary`