Mix-ME: Quality-Diversity for Multi-Agent Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

In many real-world systems, such as adaptive robotics, achieving a single, optimised solution may be insufficient. Instead, a diverse set of high-performing solutions is often required to adapt to varying contexts and requirements. This is the realm of Quality-Diversity (QD), which aims to discover a collection of high-performing solutions, each with their own unique characteristics. QD methods have recently seen success in many domains, including robotics, where they have been used to discover damage-adaptive locomotion controllers. However, most existing work has focused on single-agent settings, despite many tasks of interest being multi-agent. To this end, we introduce Mix-ME, a novel multi-agent variant of the popular MAP-Elites algorithm that forms new solutions using a crossover-like operator by mixing together agents from different teams. We evaluate the proposed methods on a variety of partially observable continuous control tasks. Our evaluation shows that these multi-agent variants obtained by Mix-ME not only compete with single-agent baselines but also often outperform them in multi-agent settings under partial observability.

1 Introduction

The conventional paradigm of optimisation has largely focused on finding a single, optimal solution that performs exceptionally well on a given problem. However, for many real-world tasks, there is need for solutions that exhibit varied behaviour across different contexts or dimensions. In such scenarios, the concept of quality-diversity [QD, 14, 8] comes into play.

QD methods aim to discover a diverse set of high-performing solutions that span different dimensions of a problem space. Unlike traditional optimisation methods that converge to a single optimal solution, QD methods produce a population of solutions that are both high-quality and diverse. This is particularly useful in problems where a single "best" solution is either not sufficient or not meaningful. For example, in robotic locomotion, there is need for strategies that adapt to malfunctions. For a robot with a damaged limb, the optimal movement pattern would differ significantly from its undamaged state. Therefore, discovering a collection of diverse gaits ensures robustness against unforeseen damages [9, 6].

The realm of multi-agent systems (MAS) presents a fertile ground for the application of QD methods. In many real-world systems, multiple agents interact in a shared environment to achieve a common goal. These systems are often partially observable, meaning that each agent has a limited view of the full state of the environment. For instance, in robotic control, there might be latency or bandwidth constraints [24, 22] that limit the amount of information that can be shared between different parts of the robot. In such cases, each body part needs to act intelligently based on its own partial information [28].

Despite clear benefits, QD has not been extensively applied to multi-agent learning. Most mainstream works in the field of multi-agent systems rely on traditional optimisation methods that do not capture

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

the essence of diversity across solutions. Furthermore, those works that do train for diversity are usually based on mutual information, making it hard to specify the type of diversity induced. Applications of MAP-Elites [19], a popular QD algorithm, to multi-agent problems have been limited to either rule-based agents [3] or environments providing dense agent-specific rewards [11], presenting a significant gap in the literature.

This paper addresses this gap by exploring how MAP-Elites can be extended to cooperative multi-agent problems, specifically in partially observable continuous control tasks. We propose *Mix-ME*, a novel extension of the MAP-Elites algorithm to the multi-agent setting. Mix-ME maintains a set of solutions and progressively refines them by generating new ones through random mutation and a crossover mechanism that mixes together agents from different teams.

We rigorously compare Mix-ME to a naive multi-agent baseline and against single-agent policies through empirical evaluation, including a sensitivity analysis on policy network size as well as generalisation experiments. This comparative analysis provides insight into the strengths and weaknesses of each approach, adding to our understanding of how open-ended learning methods can be effectively applied in various multi-agent settings [26].

2 Related Work

Single-Agent QD Quality-Diversity (QD) methods have been successfully applied to a variety of single-agent continuous control tasks. Much of this stems from the work of Cully et al. [9], who introduced the MAP-Elites algorithm and demonstrated its effectiveness for damage adaptation in robotic locomotion. Since then, MAP-Elites has seen widespread use in the robotics community, with QDax, a recent JAX-based library of QD algorithms by Lim et al. [16], enabling massive speedup on acceleration hardware. They also show that MAP-Elites can be parallelised by batching multiple grid updates in a single step, without sacrificing performance. This has brought training times down from days to minutes, making works such as this possible on a reasonable time scale. More recently, Chalumeau et al. [4] have shown that MAP-Elites and its derivatives are competitive with deep RL diversity algorithms in terms of fitness and skill discovery, despite the former being simpler and less sample-efficient. The authors tested their methods on a variety of continuous control tasks, including the unidirectional Ant, Walker2d and HalfCheetah tasks, which we also use in our experiments. The work of Allard et al. [1] bears some resemblance to ours, as they also decompose the robot's movement into separate limbs movements. Using MAP-Elites, they compute a hierarchical structure of grids, where each grid is responsible for a different level of abstraction. This parallels our approach of decomposing the robot into multiple controllers. However, they use a centralised algorithm to determine the next action and the individual leg controllers do not have policies of their own, but execute a sequence of predefined movements.

Multi-Agent QD Despite the recent success of QD methods in single-agent settings, there is limited work on applying them to multi-agent problems. Some work has been done on ad-hoc teamwork and zero-shot coordination (ZSC) in the game of Hanabi: Canaan et al. [3] use MAP-Elites with self-play to train a collection of agents, however, their agents are rule-based; ADVERSITY by Cui et al. [7] is a RL method to produce diverse teams of agents for turn-based games with public actions; and TrajeDi by Lupu et al. [17] produces diverse and robust policies for ZSC, based on a generalised Jensen-Shannon Divergence. Ridge Rider, proposed by Parker-Holder et al. [23], is a novel method for exploring the loss landscape by following the eigenvectors of the Hessian. They achieve diverse solutions effective for ZSC in a simple coordination game. Another work, by Li et al. [15], achieves diversity between agents by maximising mutual information between agents' identities and trajectories, improving on previous Google Research Football [13] and StarCraft II [25] benchmarks. Unsupervised environment design (UED) is yet another approach for achieving diversity, as shown by Samvelyan et al. [26], who use UED to design a curriculum for training a population of diverse agents for robustness in zero-sum games. Finally, a more QD-like algorithm, coupled with PPO, is used by Dixit and Tumer [11] to train a team of agents in a cooperative 2D exploration game. Their algorithm shows promising results, however, it requires dense agent-specific rewards, which are not always available in real-world scenarios.

3 Background

3.1 Quality-Diversity

Quality-diversity [QD, 14, 8] is a paradigm of evolutionary computation where the aim is to discover a diverse set of high-performing solutions that span different dimensions of a problem space. Whereas traditional optimisation methods aim to find a single solution $x \in \mathcal{X}$ that maximises an objective function $\texttt{fitness} : \mathcal{X} \mapsto \mathbb{R}$, QD methods aim to find a collection of solutions $X \subset \mathcal{X}$ where each solution $x \in X$ is high-performing in a different way. This diversity is defined in terms of a solution's behaviour descriptor (or feature vector) $\texttt{behaviour\_descriptor} : \mathcal{X} \mapsto \mathcal{B}$, that maps the solution to a vector of features that describe its behaviour, attributes or characteristics. The behaviour descriptor is a parameterisation of what kind of diversity we are interested in and is hand-crafted based on the characteristics of the problem domain.

MAP-Elites MAP-Elites [19] is one of the fundamental QD algorithms and underlies most of current research in the field. In its simplest form, MAP-Elites discretises the behaviour space into an initially empty grid X of cells with the same dimensionality as the behaviour descriptor. Each cell in the grid can hold one solution, called an *elite*. In the case of two solutions having the same behaviour descriptor, the algorithm only keeps the one with the higher fitness. Before starting the main loop, the algorithm populates the grid with $N_{\text{initial solutions}}$ random solutions. Then, each iteration proceeds by sampling a random solution x from the grid X, randomly mutating it to produce an offspring x', evaluating x' and computing its behaviour descriptor $\mathbf{b}'$. Using $\mathbf{b}'$, the algorithm looks up the relevant cell in the grid. If x' has higher fitness than the current elite in cell $\mathbf{b}'$, the elite is replaced with x'. This process is repeated for $N_{\text{iterations}}$ iterations, gradually filling the grid with high-performing solutions. The full algorithm is shown in Algorithm 1.

Algorithm 1: MAP-Elites Algorithm

Input: Initial number of solutions $N_{\text{initial solutions}}$, number of iterations $N_{\text{iterations}}$

Output: A grid X of high-performing solutions

Initialise:

- Create D-dimensional grid of solutions X and fitnesses F
- Populate the grid with $N_{\text{initial solutions}}$ random solutions.

for $i = 1$ **to** $N_{\textit{iterations}}$ **do**

- $x \leftarrow \texttt{sample\_solution}(X)$
- $x' \leftarrow \texttt{mutate}(x)$
- $f \leftarrow \texttt{fitness}(x')$
- $\mathbf{b}' \leftarrow \texttt{behaviour}(x')$
- **if** $f > F[\mathbf{b}']$ ***or*** $X[\mathbf{b}']$ *is empty* **then**
  - $X[\mathbf{b}'] \leftarrow x'$
  - $F[\mathbf{b}'] \leftarrow f$

A big advantage of MAP-Elites is that it is highly parallelisable. In practice, the algorithm is implemented by running multiple instances of the main loop in parallel. This allows for massive parallelisation, which is a big driver of the algorithm's success [16]. This counterbalances the fact that QD approaches usually require a large number of iterations to reach good solutions.

3.2 Cooperative Multi-Agent Learning

In this work, we consider partially observable cooperative multi-agent problems defined using DecPOMDP [21]. Dec-POMDP is a 7-tuple $(\mathcal{S}, \{\mathcal{A}^{(i)}\}_{i=1}^N, \mathcal{P}, r, \{\mathcal{Z}^{(i)}\}_{i=1}^N, \mathcal{O}, \gamma)$, where $\mathcal{S}$ is the set of possible states of the environment; $\mathcal{A}^{(i)}$ is the set of actions available to agent i; $\mathcal{P} : \mathcal{S} \times \mathcal{A}^{(1)} \times \cdots \times \mathcal{A}^{(N)} \times \mathcal{S} \mapsto [0, 1]$ is the transition probability function, where $\mathcal{P}(s'|s, a^{(1)}, \ldots, a^{(N)})$ is the probability of transitioning to state s' after the agents simultaneously take actions $a^{(1)}, \ldots, a^{(N)}$ in state s; $r : \mathcal{S} \times \mathcal{A}^{(1)} \times \cdots \times \mathcal{A}^{(N)} \mapsto \mathbb{R}$ is the expected reward $r = \mathbb{E}\left[R \mid s, a^{(1)}, \ldots, a^{(N)}\right]$ received after the agents take actions $a^{(1)}, \ldots, a^{(N)}$ in state s; $\mathcal{Z}^{(i)}$ is the set of observations available to agent i; $\mathcal{O} : \mathcal{S} \times \mathcal{Z}^{(1)} \times \cdots \times \mathcal{Z}^{(N)} \mapsto [0, 1]$ is the observation probability function, where $\mathcal{O}(z^{(1)}, \ldots, z^{(N)} \mid s)$ is the probability of observing $z_1, \ldots, z_N$ after transitioning to state s; $\gamma \in [0, 1]$ is the discount factor for trading off immediate and future rewards. At each time step t, every agent i receives a partial observation $z_t^{(i)}$ of the environment state s_t, and then they all independently, but simultaneously, select actions $a_t^{(1)}, \ldots, a_t^{(N)}$ based on their own policies. The environment then transitions to a new state s_{t+1} according to the transition probability

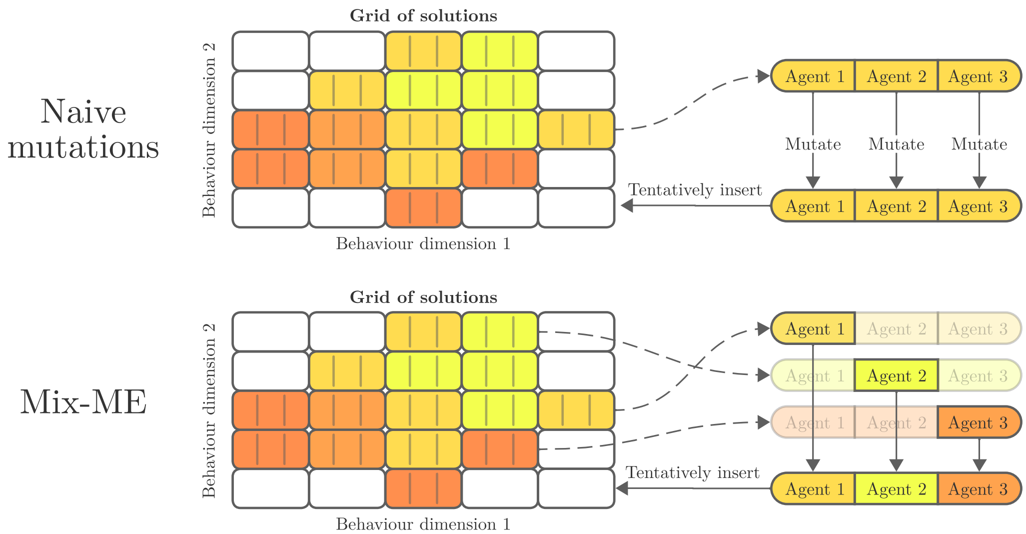


Figure 1: Graphical illustration of the difference between naive mutations and the team-mixing operation proposed in Mix-ME.

function $\mathcal{P}$, and the agents receive a joint reward r_{t+1} according to the reward function r. The goal of the agents is to learn a joint policy $\pi = (\pi_1, \ldots, \pi_N)$ that maximises the expected return $J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid \pi\right]$. Since agents do not individually have access to the full environment state, they must learn to collaborate in order to achieve the goal.

4 Methods

In this section, we present the design of the proposed multi-agent QD approaches. All of the methods described below are based on the same core MAP-Elites algorithm, which is described in Algorithm 1. However, we assume two main changes to the definitions:

1. The parameter space $\mathcal{X}$ is now a set of N parameter spaces $\mathcal{X}_1, \ldots, \mathcal{X}_N$, one for each agent.
2. Solutions in the grid are now tuples $(x_1, \ldots, x_N)$, where x_i is the solution for agent i.

The first change is necessary since agents can have different action and observation spaces, and therefore need to be sampled from different parameter spaces. The second change is needed to allow the algorithm to keep track of each individual agent policy.

4.1 Naive Multi-Agent MAP-Elites

The most straightforward way to train groups of cooperative agents with MAP-Elites is to treat the group as a single unit, and use the single-agent variation of MAP-Elites. In this approach, a new offspring is created by sampling a random solution $(x_1, \ldots, x_N)$ from the grid and then mutating each of the agents' policies x_i to produce a new policy x'_i. The resulting team of agents $(x'_1, \ldots, x'_N)$ is then evaluated and assigned to the grid. The algorithm is illustrated in Figure 1**(top)**. The mutations we use are polynomial mutation [10] and isoline variation [Iso-LineBB, 30].

This approach is simple and easy to implement, but its restrictive sampling strategy can limit its potential. The fact that every agent in the offspring is derived from the same parent solution means that the algorithm is not able to combine policies from different parents. This might lose out on some potential benefits of co-adaptation between agents.

4.2 Mix-ME

One relaxation of the baseline approach is to allow the agents in the offspring to be derived from different parents. In a multi-agent system, different agents might have specialised roles that require different capabilities or expertise. The motivation for this approach is that during training, we might have multiple solutions in the grid containing agents that are proficient in different roles. By allowing agents in an offspring to inherit policies from different parents, the algorithm can combine experts from different teams and therefore promote the co-adaptation of agents with complementary roles. An analogy to this approach is the formation of sports teams, where the coach might select a strong goalkeeper from one team, a strong striker from another team, and so on.

We thus introduce the *Agent Mixing MAP-Elites* (Mix-ME), a novel multi-agent QD approach that performs mix-and-matching of individual agents between distinct groups within the grid. In Mix-ME, in addition to using naive mutations on raw parameters, it includes a team-crossover operator. This operator creates a new offspring by sampling N random solution tuples $\left(x_1^{(1)}, \ldots, x_N^{(1)}\right), \ldots, \left(x_1^{(N)}, \ldots, x_N^{(N)}\right)$ with replacement from the grid. The offspring is then created by taking the 1st agent from the 1st tuple, the 2nd agent from the 2nd tuple, and so on. The resulting team of agents $\left(x_1^{(1)}, \ldots, x_N^{(N)}\right)$ is then evaluated and assigned to the grid. This operator is illustrated in Figure 1**(bottom)**.

Since massive parallelism in MAP-Elites is achieved by producing new solutions in batches, in each iteration, we split the batch evenly across operators. Thus, with a batch size of N_{batch}, $N_{\text{batch}}/3$ new solutions would be formed using polynomial mutation, $N_{\text{batch}}/3$ with isoline variation, and $N_{\text{batch}}/3$ using team-crossover. The purpose of the conventional mutation operators is to optimise the weights, resulting in local hill-climbing behaviour, while the purpose of the team-crossover operator is to promote co-adaptation of agents with complementary roles.

5 Experimental Setup

In this section, we explain the motivation, design and setup for our experiments, as well as describing the training environments. The main questions we seek to answer are:

1. How do the proposed algorithms compare against each other and against the single-agent baseline in terms of performance, diversity and generalisation capability?
2. Do specific traits of the environment affect the performance of the proposed algorithms?
3. How does changing the size of the policy networks impact the performance of the different MAP-Elites methods?

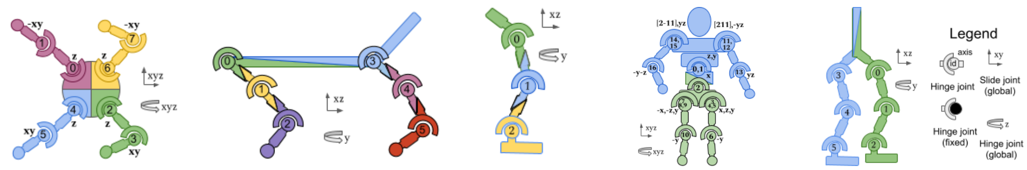
For details on our experimental implementation and hyperparameter settings, please refer to Appendices A.2 and A.3.

Environments To evaluate the proposed methods, we extend five existing single-agent continuous control environments in the Brax physics engine [12] to support multiple agents. These environments are the multi-agent parallels of the single-agent Mujoco environments [29] and were first introduced by Peng et al. [24]. Previous implementations, however, have not natively supported JAX [2], and their parallelisability has been limited. Our implementation uses pure JAX and is highly parallelisable, allowing for massive speedup on acceleration hardware. Moreover, it is compatible with the QDax library [16, 5] which we base our QD algorithms on.

The environments adapt the single-agent MuJoCo tasks to multi-agent use with the concept of factored robots. In this paradigm, the robot is partitioned into multiple components, each controlled by an individual agent. Figure 2 illustrates the factorisation for each environment. These agents have partial observability of the global state and act based on local information. They must then collaboratively control the robot to accomplish the task. The environment specifics are described in Table 1, and in more detail in Appendix A.1.

The behaviour descriptor we use for all environments is the average time that each foot of the robot is in contact with the ground during an episode, represented by

$$\mathbf{b} = \frac{1}{T}\sum_{t=1}^{T}\begin{pmatrix}\mathbb{I}[\text{foot 1 touches ground}]\\ \mathbb{I}[\text{foot 2 touches ground}]\\ \vdots \\ \mathbb{I}[\text{foot N touches ground}]\end{pmatrix} \tag{1}$$



(a) Ant (b) HalfCheetah (c) Hopper (d) Humanoid (e) Walker2D

Figure 2: Illustration of the robot factorisations. The colours represent the different agents. Image sourced from Peng et al. [24].

Table 1: Summary of the environments used in our experiments.

Environment	**Agents**	**Observation Space**		**Action Space**	
		Single-Agent	Multi-Agent	Single-Agent	Multi-Agent
Ant	4	28	(18 each)	8	(2 each)
HalfCheetah	6	18	(9, 9, 8, 8, 9, 8)	6	(1 each)
Hopper	3	11	(8, 9, 8)	3	(1 each)
Humanoid	2	376	(248, 176)	17	(9, 8)
Walker2D	2	22	(17 each)	6	(3 each)

where T is the length of the episode, and $\mathbb{I}$ is the indicator function. This behaviour descriptor is simple but effective for capturing various gaits of the robot. For example, a hopping gait would have a low value for all feet, while a walking gait would have a higher value. It has also been used in previous studies [9] to allow robots to recover from mechanical damage.

Evaluation Metrics We use the following three metrics when comparing the performances of baselines. Firstly, the *maximum fitness* $f_{\text{max}} = \max_{f \in F} f$, i.e. the fitness of the best performing solution in the grid at the end of training, where fitness refers to the total reward received during an episode. Secondly, the *coverage* $C = \frac{\text{number of cells containing a solution}}{\text{total number of cells}}$, representing the proportion of the behaviour space that solutions have been found for. The coverage is a measure of the diversity of the solution grid. Thirdly, we measure the *QD score*, $QD = \sum_{f \in F} f$, the sum of the fitnesses of all solutions in the grid at the end of training. This score summarises both performance and diversity of the solution grid and is the main metric we use to compare MAP-Elites methods against each other.

Generalisation Experiments To assess the generalisation capabilities of our proposed algorithms [18], we follow the experimental procedure outlined in a previous paper by Chalumeau et al. [4]. This procedure employs a few-shot adaptation approach in modified environments, where pre-computed policies are evaluated without retraining. We explore two distinct settings: gravity update and leg dysfunction. In the gravity update scenario, the gravity constant is modified by multiplying it with a coefficient over a specified range. In the leg dysfunction setting, we alter the input-to-torque coefficients of a single leg across a range.

Initially, each baseline is trained for 1,000 iterations in a standard environment. Then, we conduct 100 evaluations for each solution in the grid using the modified environments, calculating the median fitness for each solution. The maximum of these median fitness values is then reported. To ensure robustness and reliability, we report the results of the experiments across 10 different seed values.

6 Results and Discussion

6.1 Comparison of Multi-Agent MAP-Elites Methods

Performance and Diversity We first compare the performance of the naive multi-agent MAP-Elites baseline, Mix-ME, and the single-agent baseline. Figure 3 illustrates the learning curves for each of the environments. We can observe several interesting trends.

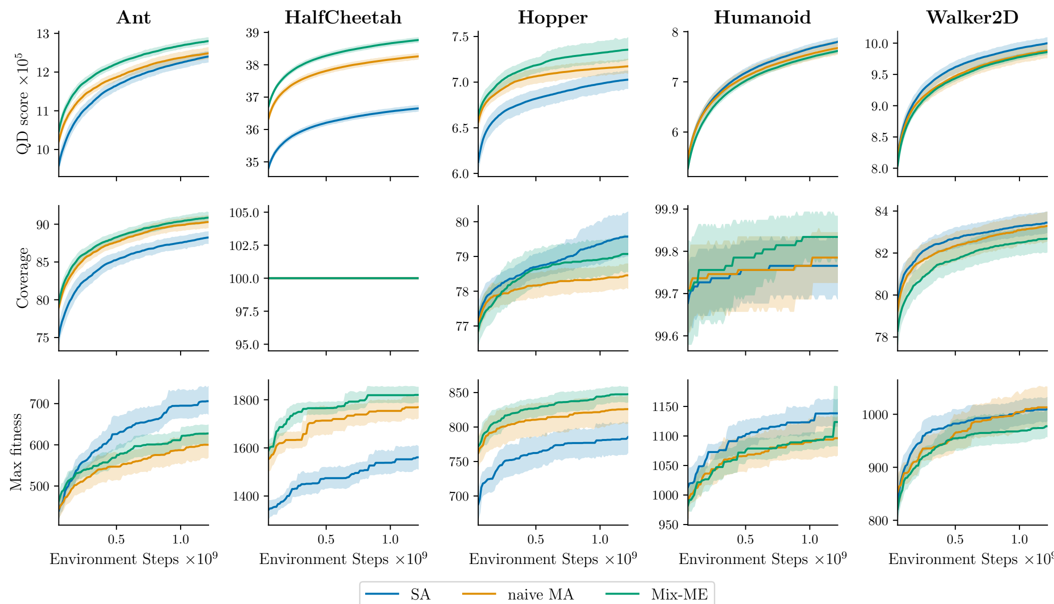


Figure 3: Learning curves of the multi-agent (Naive MA & Mix-ME) and single agent (SA) MAP-Elites methods. The shaded regions represent the standard deviation across 10 runs. On the x-axis, we show the total number of environment steps taken by the algorithm, which is equal to the number of iterations multiplied by the number of offspring per iteration.

First, we see that across environments with more than 2 agents (Ant, HalfCheetah, Hopper), Mix-ME consistently outperforms the naive multi-agent baseline on every metric. On the other hand, in the Walker2d environment, we see the opposite trend, where the naive baseline outperforms Mix-ME, only slightly in terms of QD score, but significantly in terms of coverage and maximum fitness. This suggests that mixing elites builds better teams by exploiting diversity in the solution grid, but loses its effectiveness when the number of agents is small.

Another interesting observation is that in terms of QD score, both multi-agent methods outperform the single-agent baseline in environments with more than 2 agents. In fact, if we look at the score as a function of the number of agents, we see that the performance gap roughly increases with the number of agents, as shown in Figure 4. An important caveat, however, is that the multi-agent methods have the same policy network architecture per agent as the single-agent baseline, which means that the total number of parameters in the multi-agent methods is comparably larger. We explore the effect of policy network size in Section 6.2 and show that this fact alone does not explain the difference in performance. Therefore, the multi-agent methods are indeed able to learn a higher-performing solution grid.

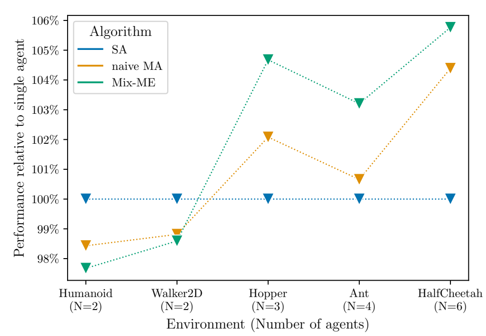


Figure 4: Performance (QD score) at the end of training, as a percentage relative to the single-agent baseline, ordered by number of agents.

Apart from performance, in Appendix A.4 we also show the resulting solution grids for each environment. We do not see any obvious differences between the grids of different methods, except for the Ant environment, where the solution grid for the multi-agent methods seems to be more uniform than the single-agent baseline. This is consistent with Figure 3, which shows higher QD-scores but lower maximum fitness for the multi-agent methods than the single-agent baseline. Partial observability might be a factor here, since the agents only have access to local information, and therefore might not be able to learn as high-performing policies as the single-agent baseline.

Generalisation We also evaluate the generalisation capabilities of the different methods in the leg dysfunction and gravity update scenarios. The results are shown in Figure 5 and show similar trends to the results in the previous section. We see that Mix-ME generalises better than the naive multi-agent baseline in environments with more than 2 agents, but worse in the Walker2d environment.

We also see that in the leg dysfunction scenario, the performance of the single-agent baseline drops significantly faster than the multi-agent methods as the test environment diverges from the training environment. In the gravity update scenario, the results are mixed and highly dependent on the environment. A key difference between the two scenarios is that leg dysfunction is exactly the failure mode that the behaviour descriptor is designed to capture; the descriptor is a parameterisation of each leg's contact time with the ground, and therefore the resulting grid of solutions should contain solutions that are diverse in terms of individual leg usage. On the other hand, it is not obvious how this kind of diversity would be useful in the gravity update scenario.

These results have a straightforward interpretation: since multi-agent methods learn higher-performing and more diverse solutions in environments with many agents, if the behaviour descriptor is well-aligned with the task, naturally the multi-agent methods will be able to learn more robust solutions that are less sensitive to changes in the environment, compared to their single-agent counterpart.

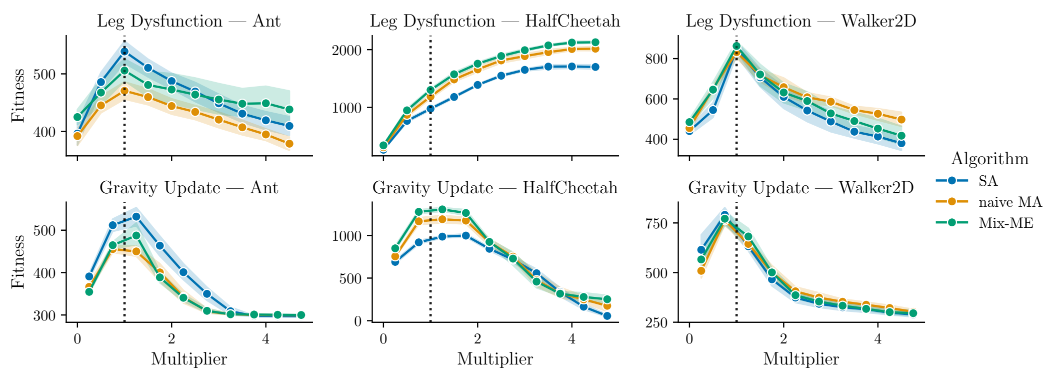


Figure 5: Generalisation results. The x-axis shows the multiplier applied to the gravity constant in the gravity update scenario, and the coefficient applied to the input-to-torque coefficients in the leg dysfunction scenario. The y-axis shows the median fitness of the best solution in the grid.

6.2 Effect of Policy Network Size

When comparing single-agent and multi-agent baselines, one subtle caveat is that each individual agent's policy network in the multi-agent methods has the same architecture as the single-agent baseline has for controlling the entire robot. As a result, the total number of parameters in the multi-agent methods is N_{agents} times larger than the single-agent baseline. To address this issue, we conduct an experiment where we vary the size of the policy networks in each method, and observe how the performance scales. Note that we performed hyperparameter tuning for each policy network size separately to ensure optimal learning.

Figure 6 shows the results of this experiment. We can see that in none of the environments does increasing the policy network size of the single-agent baseline result in comparatively better performance than the multi-agent methods with smaller policy networks. In other words, we don not gain much by increasing the policy network size of the single-agent baseline. In fact, in most environments, the performance either drops or stays the same with increasing policy network size.

This, in combination with the results from Section 6.1, means that the good performance of the multi-agent methods cannot simply be attributed to the larger number of parameters. Instead, it suggests that there must be some benefit to learning a decentralised policy, even though this imposes partial observability on each agent.

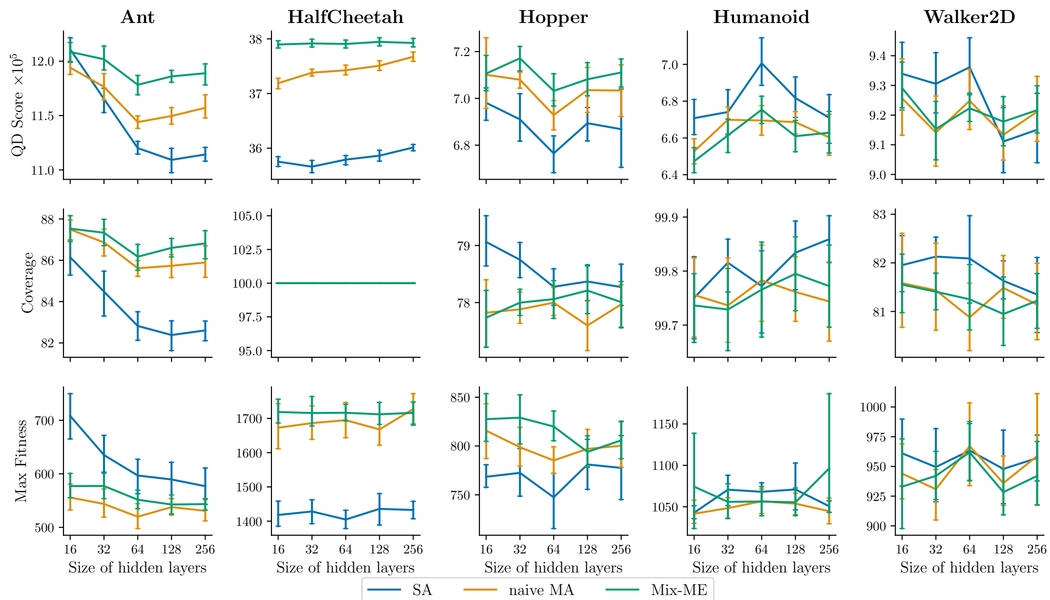


Figure 6: Effect of policy network size on performance of the different methods. The x-axis shows the number of units in each of the two hidden layers. 95% confidence intervals are shown as error bars. Note that we have reduced the batch size here to 1024 to allow for larger networks, meaning absolute performance is not comparable to previous sections.

7 Conclusion and Future Work

This work sets out to bridge the gap between QD and cooperative multi-agent learning. It was motivated by the observation that many real-world continuous control tasks are inherently partially observable and multi-agent, and that often, we are interested in inducing diversity in the solutions to these tasks, yielding a set of high-quality solutions, that are robust to damage and to changes in the environment.

To this end, we proposed Mix-ME, a new multi-agent variant of the MAP-Elites algorithm, which adds a team-crossover operation to form new solutions. We presented a comprehensive set of experiments that compare the performance of our proposed method against a naive multi-agent extension and against a single-agent baseline. These experiments revealed that Mix-ME shows superior performance and generalisation capabilities, and that this performance gap increases with the number of agents. We also showed that in many-agent environments, decentralised control policies trained using Mix-ME outperform single-agent policies trained using normal MAP-Elites, even under partial observability.

There are numerous avenues for future work. First, benchmarking Mix-ME on different environments would be a good way to further validate the results. This paper only includes continuous control environments with a relatively small number of agents, and it would be interesting to see how the methods scale to environments with more agents. Environments with discrete action spaces, such as grid-worlds, would also be beneficial to explore. Another avenue to explore is multi-agent extensions of more sophisticated MAP-Elites variants, such as Policy gradient assisted MAP-Elites [PGA-ME, 20], which employs first-order optimisation techniques. This could potentially lead to better scaling and performance, making MAP-Elites methods more competitive with policy gradient methods in terms of maximum fitness. Our proposed methods extend easily to these variants, and therefore could be a good starting point for future work.

This paper has shown that MAP-Elites methods are a promising approach to inducing diversity in multi-agent learning. We hope that this work will inspire further research in this direction, and that it will help to bridge the gap between QD and cooperative multi-agent learning.

References

[1] Maxime Allard, Simón C. Smith, Konstantinos Chatzilygeroudis, and Antoine Cully. Hierarchical quality-diversity for online damage recovery. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, page 58–67, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392372. doi: 10.1145/3512290.3528751. URL `https://doi.org/10.1145/3512290.3528751`. 2

[2] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs. `http://github.com/google/jax`, 2018. URL `http://github.com/google/jax`. 5

[3] Rodrigo Canaan, Julian Togelius, Andy Nealen, and Stefan Menzel. Diverse agents for ad-hoc cooperation in hanabi, 2019. 2

[4] 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 *International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=6BHlZgyPOZY`. 2, 6

[5] Felix Chalumeau, Bryan Lim, Raphael Boige, Maxime Allard, Luca Grillotti, Manon Flageat, Valentin Macé, Arthur Flajolet, Thomas Pierrot, and Antoine Cully. Qdax: A library for quality-diversity and population-based algorithms with hardware acceleration. *arXiv preprint arXiv:2308.03665*, 2023. 5

[6] 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*. ACM, jun 2020. doi: 10.1145/3377930.3390217. 1

[7] Brandon Cui, Andrei Lupu, Samuel Sokota, Hengyuan Hu, David J Wu, and Jakob Nicolaus Foerster. Adversarial diversity in hanabi. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=uLE3WF3-H_5`. 2

[8] Antoine Cully and Yiannis Demiris. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22(2):245–259, 2018. doi: 10.1109/TEVC.2017.2704781. 1, 3

[9] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015. doi: 10.1038/nature14422. URL `https://doi.org/10.1038/nature14422`. 1, 2, 6, 15

[10] Kalyanmoy Deb and Samir Agrawal. A niched-penalty approach for constraint handling in genetic algorithms. In *Artificial Neural Nets and Genetic Algorithms*, pages 235–243, Vienna, 1999. Springer Vienna. ISBN 978-3-7091-6384-9. 4

[11] Gaurav Dixit and Kagan Tumer. Balancing teams with quality-diversity for heterogeneous multiagent coordination. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, page 236–239, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392686. doi: 10.1145/3520304.3529062. URL `https://doi.org/10.1145/3520304.3529062`. 2

[12] 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. `http://github.com/google/brax`, 2021. URL `http://github.com/google/brax`. 5

[13] Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zając, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, and Sylvain Gelly. Google research football: A novel reinforcement learning environment, 2020. 2

[14] Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011. 1, 3

[15] Chenghao Li, Tonghan Wang, Chengjie Wu, Qianchuan Zhao, Jun Yang, and Chongjie Zhang. Celebrating diversity in shared multi-agent reinforcement learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 3991–4002. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper_files/paper/2021/file/20aee3a5f4643755a79ee5f6a73050ac-Paper.pdf`. 2

[16] Bryan Lim, Maxime Allard, Luca Grillotti, and Antoine Cully. Accelerated quality-diversity for robotics through massive parallelism. *arXiv preprint arXiv:2202.01258*, 2022. 2, 3, 5

[17] Andrei Lupu, Brandon Cui, Hengyuan Hu, and Jakob Foerster. Trajectory diversity for zero-shot coordination. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 7204–7213. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/lupu21a.html`. 2

[18] Anuj Mahajan, Mikayel Samvelyan, Tarun Gupta, Benjamin Ellis, Mingfei Sun, Tim Rocktäschel, and Shimon Whiteson. Generalization in cooperative multi-agent systems. *arXiv preprint arXiv:2202.00104*, 2022. 6

[19] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *ArXiv*, abs/1504.04909, 2015. URL `https://api.semanticscholar.org/CorpusID:14759751`. 2, 3

[20] Olle Nilsson and Antoine Cully. Policy gradient assisted map-elites. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '21, page 866–875, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383509. doi: 10.1145/3449639.3459304. URL `https://doi.org/10.1145/3449639.3459304`. 9

[21] Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. Springer Publishing Company, Incorporated, 1st edition, 2016. ISBN 3319289276. 3

[22] Sylvie C. W. Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research*, 29 (8):1053–1068, 2010. doi: 10.1177/0278364910369861. URL `https://doi.org/10.1177/0278364910369861`. 1

[23] Jack Parker-Holder, Luke Metz, Cinjon Resnick, Hengyuan Hu, Adam Lerer, Alistair Letcher, Alexander Peysakhovich, Aldo Pacchiano, and Jakob Foerster. Ridge rider: Finding diverse solutions by following eigenvectors of the hessian. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 753–765. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/file/08425b881bcde94a383cd258cea331be-Paper.pdf`. 2

[24] Bei Peng, Tabish Rashid, Christian A. Schroeder de Witt, Pierre-Alexandre Kamienny, Philip H. S. Torr, Wendelin Böhmer, and Shimon Whiteson. Facmac: Factored multi-agent centralised policy gradients, 2020. 1, 5, 6

[25] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge, 2019. 2

[26] Mikayel Samvelyan, Akbir Khan, Michael D Dennis, Minqi Jiang, Jack Parker-Holder, Jakob Nicolaus Foerster, Roberta Raileanu, and Tim Rocktäschel. MAESTRO: Open-ended environment design for multi-agent reinforcement learning. In *International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=sKWlRDzPfd7`. 2

[27] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018. 13

[28] Keiki Takadama, Shuichi Matsumoto, Shinichi Nakasuka, and Katsunori Shimohara. A reinforcement learning approach to fail-safe design for multiple space robots—cooperation mechanism without communication and negotiation schemes. *Advanced Robotics*, 17:21 – 39, 2003. URL `https://api.semanticscholar.org/CorpusID:44912233`. 1

[29] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109. 5

[30] Vassiiis Vassiliades and Jean-Baptiste Mouret. Discovering the elite hypervolume by leveraging interspecies correlation. *Proceedings of the Genetic and Evolutionary Computation Conference*, Jul 2018. doi: 10.1145/3205455.3205602. URL `http://dx.doi.org/10.1145/3205455.3205602`. 4

A Appendix

A.1 Environments

Ant The Ant environment [27] is a 3-dimensional 4-legged robot with 8 rotors. We factorise it into 4 agents, each controlling the two joints on one leg. The agents observe the angle and angular velocity of the local leg joints and immediately adjacent joints, as well as the global position and velocity of the robot central body.

The goal is to make the Ant walk forward as fast as possible, while minimising energy consumption and external contact forces. All agents receive a shared reward at each time step, defined as

$$r := r_{\text{survive}} + r_{\text{forward}} - r_{\text{ctrl}} - r_{\text{contact cost}} \tag{2}$$

where r_{survive} is a constant reward for surviving, r_{forward} is the forward velocity of the robot, r_{ctrl} is a penalty for large control inputs, and $r_{\text{contact cost}}$ is a penalty for external contact forces.

HalfCheetah The HalfCheetah environment is a 2-dimensional 2-legged robot with 6 rotors. We factorise it into 6 agents, each controlling one joint. The agents observe the angle and angular velocity of their assigned joint and immediately adjacent joints, as well as the global position and velocity of the tip of the robot.

The goal is to make the HalfCheetah run forward as fast as possible, while minimising energy consumption. All agents receive a shared reward at each time step, defined as

$$r := r_{\text{forward}} - r_{\text{ctrl}} \tag{3}$$

where the individual reward components are the same as in the Ant environment.

Hopper The Hopper environment is a 2-dimensional 1-legged robot with 3 rotors. We factorise it into 3 agents, each controlling one joint. The agents observe the angle and angular velocity of their assigned joint and immediately adjacent joints, as well as the global position and velocity of the top of the robot.

The goal is to hop forward as fast as possible, while minimising energy consumption. All agents receive a shared reward at each time step, defined as

$$r := r_{\text{survive}} + r_{\text{forward}} - r_{\text{ctrl}} \tag{4}$$

where the individual reward components are the same as in the Ant environment.

Humanoid The Humanoid environment is a 3-dimensional 2-legged robot with 20 rotors, designed to resemble a human. We factorise it into 2 agents, one controlling the upper body and the other controlling the lower body. The agents observe the angle and angular velocity of their assigned joints and immediately adjacent joints, as well as the global position and velocity of the humanoid's torso.

The goal is to make the Humanoid walk forward as fast as possible, while minimising energy consumption. All agents receive a shared reward at each time step, defined as

$$r := r_{\text{survive}} + r_{\text{forward}} - r_{\text{ctrl}} \tag{5}$$

where the individual reward components are the same as in the Ant environment.

A.2 Implementation Details

In each of our experiments, we perform 10 runs with different random seeds and report the mean and standard deviation of the results. Each run consists of 1000 iterations of the MAP-Elites algorithm, where each iteration produces 4096 offspring. We evaluate offspring in parallel for 300 timesteps on a single GPU for each job. We use GPUs of types NVIDIA GTX 1080 Ti, RTX 2080 Ti, Tesla P100, V100, and A100.

A.3 Hyperparameters

In order to ensure a fair comparison between the different methods, we tuned mutation hyperparameters for each combination of environment and policy network size. The hyperparameters were tuned by running a grid search over a range of values for each hyperparameter, and selecting the combination that yielded the highest QD score averaged over 3 seeds. These optimal hyperparameters were then used for all experiments. We used a fully connected multi-layer perceptron with 2 hidden layers of 64 units each, save for the policy network sensitivity analysis where the hidden layer size was modified.

Table 2: Search space for MAP-Elites mutation hyperparameters.

Hyperparameter	**Search space**
σ_{iso}	$\{0.0001, 0.001, 0.01, 0.1, 1.0\}$
σ_{line}	$\{0.0001, 0.001, 0.01, 0.1, 1.0\}$
η	$\{4, 8, 16, 32, 64, 128, 256\}$

Table 3: Optimal MAP-Elites hyperparameters for each environment and policy network size.

Environment	**Policy network hidden layer size**	σ_{iso}	σ_{line}	η
Ant	16	0.001	1.0	32
	32	0.001	1.0	64
	64	0.001	1.0	128
	128	0.001	1.0	128
	256	0.001	1.0	128
HalfCheetah	16	0.01	0.1	128
	32	0.001	0.1	128
	64	0.001	0.1	128
	128	0.001	0.1	128
	256	0.001	0.1	128
Hopper	16	0.001	0.1	8
	32	0.001	0.1	16
	64	0.001	0.1	16
	128	0.001	0.1	64
	256	0.001	0.1	128
Humanoid	16	0.001	1.0	32
	32	0.001	1.0	64
	64	0.001	1.0	128
	128	0.001	1.0	128
	256	0.001	1.0	128
Walker2d	16	0.001	0.1	4
	32	0.001	0.1	4
	64	0.01	0.1	8
	128	0.001	0.1	8
	256	0.01	0.01	8

A.4 MAP-Elites Behaviour Descriptor Grid

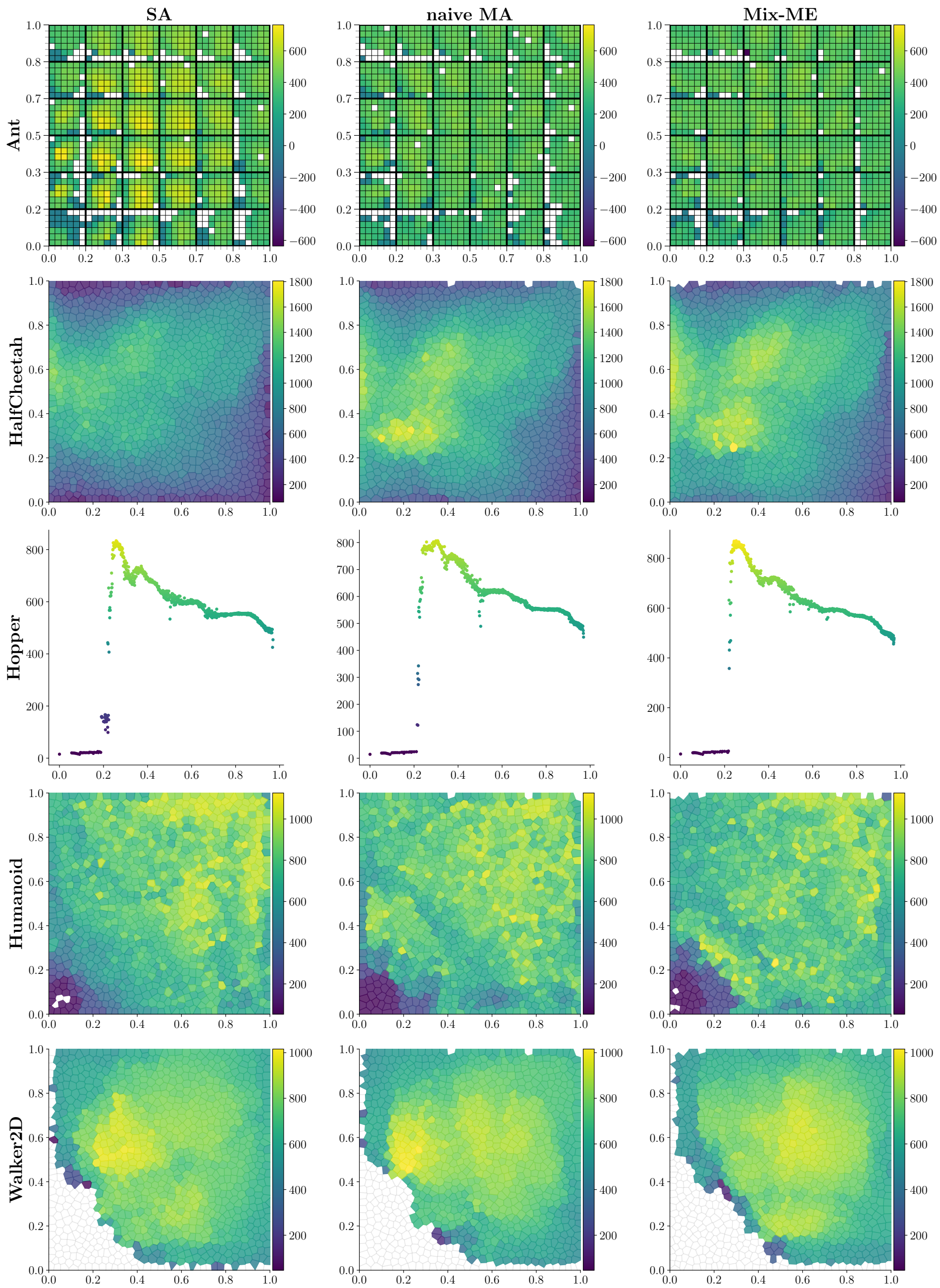


Figure 7: Visualisation of the solution grids produced by the different multi-agent MAP-Elites methods, broken down by environment. The visualisation for the 4-dimensional descriptor in the Ant environment is projected into 2D as is done in Cully et al. [9].