Relevance-guided Unsupervised Discovery of Abilities with Quality-Diversity Algorithms

Luca Grillotti  
Adaptive and Intelligent Robotics Lab  
Imperial College London, United Kingdom  
luca.grillotti16@imperial.ac.uk

Antoine Cully  
Adaptive and Intelligent Robotics Lab  
Imperial College London, United Kingdom  
a.cully@imperial.ac.uk

ABSTRACT
Quality-Diversity algorithms provide efficient mechanisms to generate large collections of diverse and high-performing solutions, which have shown to be instrumental for solving downstream tasks. However, most of those algorithms rely on a behavioural descriptor to characterise the diversity that is hand-coded, hence requiring prior knowledge about the considered tasks. In this work, we introduce Relevance-guided Unsupervised Discovery of Abilities; a Quality-Diversity algorithm that autonomously finds a behavioural characterisation tailored to the task at hand. In particular, our method introduces a custom diversity metric that leads to higher densities of solutions near the areas of interest in the learnt behavioural descriptor space. We evaluate our approach on a simulated robotic environment, where the robot has to autonomously discover its abilities based on its full sensory data. We evaluated the algorithms on three tasks: navigation to random targets, moving forward with a high velocity, and performing half-rolls. The experimental results show that our method manages to discover collections of solutions that are not only diverse, but also well-adapted to the considered downstream task.

CCS CONCEPTS
• Computer systems organization → Embedded systems; Redundancy; Robotics; • Networks → Network reliability.

KEYWORDS
Quality-Diversity Optimisation, Unsupervised Learning, Robotics

1 INTRODUCTION
One of the motivations of using learning algorithms in robotics is to enable robots to discover on their own how to solve a task. Ideally, a robot should be able to discover on its own how to manipulate new objects [22] or how to adapt its behaviours when facing an unseen situation like a mechanical damage [10]. However, despite many impressive breakthroughs in learning algorithms for robotics applications over the last decade [1, 20], these methods still require a significant amount of engineering to become effective, for instance, to define reward functions, or characterise the expected behaviours.

An attempt to reduce the amount of engineering required to discover the skills of a robot has been proposed with the AU- RORA algorithm [9, 18]. This algorithm leverages the creativity of Quality-Diversity (QD) optimisation algorithms to generate a collection of diverse and high-performing behaviours, called a behavioural repertoire. QD algorithms usually require the definition of a manually-defined Behavioural Descriptor (BD) to characterise the different types of behaviours contained in the repertoire. Instead, AURORA uses dimensionality-reduction techniques to perform the behavioural characterisation in an unsupervised manner. The resulting algorithms enable robots to autonomously discover a large diversity of behaviours without requiring a user to define a fitness function or a behavioural descriptor.

Nevertheless, the behavioural repertoires are often meant to be used by other processes to solve complex tasks (called downstream tasks). For instance, a behavioural repertoire containing a diversity of locomotion primitives can be used by a planning algorithm to

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

GECCO ’22, July 9–13, 2022, Boston, MA, USA © 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM. ACM ISBN 978-1-4503-9237-2/22/07 . . . $15.00 https://doi.org/10.1145/3512290.3528837

77
chain these primitives and solve complex navigation tasks [7, 13].
Yet, for this process to be effective, the repertoire needs to contain
the appropriate behaviours to solve the task at hand. AURORA is
task agnostic and aims at covering all the possible behaviours of
a robot. However, such a range can cover relevant but also irrelevant
behaviours. Moreover, given the fact that repertoires usually con-
tain a finite number of behaviours, it is crucial to ensure that most
of their capacity is used to capture behaviours that are relevant for
the considered tasks.

In this paper, we introduce Relevance-guided Unsupervised Dis-
covey of Skills (RUDA), an extension of AURORA, which not only
automatically defines the BD, but ensures that it is tailored to the
task at hand. This results in behavioural repertoires containing a
higher density of behaviours in the regions of the BD space that are
relevant for the task. We evaluated RUDA on a simulated hexapod
robot with three tasks: navigation to random targets, moving for-
ward with a high velocity, and performing half-rolls. Furthermore,
we compare RUDA to four baselines, including AURORA as well as
different ways to manually define the BD. The experimental results
show that RUDA manages to discover collections of behaviours
that are not only diverse, but also well-adapted to the considered
downstream task.

2 BACKGROUND

2.1 Quality-Diversity Algorithms

Quality-Diversity (QD) algorithms are a subclass of evolutionary
algorithms that aim at finding a container of both diverse and high-
performing individuals. In addition to standard evolutionary algo-
rithms, QD algorithms consider a BD, which is a low-dimensional
vector that characterises the behaviour of an individual. QD algo-
rithms use the BDs to quantify the novelty of an individual with
respect to the solutions already in the container. The container of
individuals is produced by the QD algorithm in an iterative manner:
(1) individuals are selected from the container and undergo random
variations (e.g. mutations, cross-overs); (2) their fitness score and
BD are evaluated; and (3) we try to add them back to the container.
If they are novel enough compared to solutions that are already in
the container, they are added to the container. If they are better than
similar individuals in the container, they replace these individuals.

The literature [6, 11] consider two types of containers: 1) grid-
based containers introduced by the MAP-Elites algorithm [33],
and 2) unstructured-archive, introduced by the Novelty-Search
algorithm [29]. In unstructured containers [11], we compare the
distance of the individual to all individuals present in the container,
using the Euclidean distance metric in the BD space. If that distance
is shorter than a threshold \( d_{\text{min}} \), then the individual is added to
the container. Otherwise, if the distance to the closest individual is
inferior to \( d_{\text{min}} \), then their respective novelty and fitness scores are
considered. If the novelty and fitness scores of the new individual
e-dominate the ones of the old individual, then the new individual
replaces the closest individual present in the container.

The minimal distance threshold \( d_{\text{min}} \) may be fixed and hand-
coded. It can also be variable and controlled to keep the size of the
container around a target value \( N_{\text{C}}^{\text{target}} \). To that end, the Container-
Size-Control (CSC) method (introduced in AURORA [18]) con-
stantly adjusts the value of the \( d_{\text{min}} \) depending on the current
container size \( C \): if \( |C| > N_{\text{C}}^{\text{target}} \), then \( d_{\text{min}} \) is increased to limit the
amount of new individuals to the container; and if \( |C| < N_{\text{C}}^{\text{target}} \),
then \( d_{\text{min}} \) is decreased to increase the amount of new individuals
added to the container. If the container presents too many individ-
uals, then some of them need to be removed, in order to maintain
the container size around \( N_{\text{C}}^{\text{target}} \). To do so, at regular iterations
\( T_{\text{C}} \), all individuals are removed from the container, and re-added
with the up-to-date distance threshold \( d_{\text{min}} \).

2.2 Discovering Unsupervised Behaviours

Quality-Diversity algorithms are a promising tool to generate a
large diversity of behaviours in robotics. However, the definition
of the BD space might not always be straightforward when the
robot or its capabilities are unknown. AURORA is a QD algorithm
designed to discover the abilities of robots, by maximising the diver-
sity of behaviours contained in the repertoire in an unsupervised
manner. AURORA automatically defines the BD by encoding the
high-dimensional sensory data generated by the robot during a
controller execution into low-dimensional representations. The
encoding can be achieved using any dimensionality reduction tech-
nique, such as Principal Component Analysis [9], and Auto-Encoder
(AE) [18].

To generate a behavioural repertoire, AURORA alternates be-
tween QD phases and Encoder phases. During the QD phase, indi-
viduals undergo the same process as in standard QD algorithms:
selection from the container, evaluation, and attempt to add to
the container. However, the evaluation is performed in a slightly
different way: instead of a low-dimensional BD, the QD task
returns high-dimensional sensory data, that is then encoded using
the dimensionality reduction technique.

During the Encoder phase, the dimensionality reduction struc-
ture (e.g. the AE) is trained using the high-dimensional data from all
the individuals present in the container. Once the encoder is trained,
the unsupervised BDs are recomputed with the new encoder for all
individuals present in the container.

Alternating between these two phases enables AURORA to pro-
gressively build its behavioural repertoires by discovering new
solutions, while refining its encoding of the high-dimensional sen-
sory data generated by the robot every time new solutions are
added to the archive.

3 RELATED WORKS

3.1 Leveraging Behavioural Repertoires to Solve
Tasks

One of the direct applications of behavioural repertoires in ro-
botics is to use them to solve downstream tasks, such as maze
navigation [7, 13], damage recovery [10, 23], or throwing objects to
specific locations [21, 24]. For example, Chatzilygeroudis et al. [7]
introduced the Reset-free Trial and Error (RTE) algorithm. This algo-
rithm leverages a repertoire of locomotion controllers that enable
a hexapod robot to walk in various directions at different speeds.
RTE then uses this set of behaviours as primitive actions with a
planning algorithm (Monte Carlo Tree Search [8]) to execute com-
plex trajectories. RTE is also coupled with a Gaussian process that
predicts the effect of each primitive action and is updated after each
action execution. Thanks to these predictions, RTE enables the robot to solve its maze navigation task while crossing the reality-gap (including damage-recovery and environmental changes).

Another example of behavioural repertoire usage is the APROL algorithm [23], which creates a collection of behavioural repertoires to improve the resilience of robots to damages and their ability to generalise to different objects. Each repertoire is generated by considering a different condition. For example, a different damaged leg on a hexapod robot in the case of a locomotion task, or an object with a different shape in the context of object manipulation. Then during deployment, APROL searches within its collection of repertoires the one that resembles the most to the current situation and then uses the behaviours it contains to solve the task. This automatic repertoire selection enables APROL to execute behaviours that are more appropriate for the current situation, while being able to cover a larger range of possible scenarios thanks to the options provided by the collection of repertoires.

However, deciding on the most appropriate definitions for the BD and the fitness for a given task requires a certain level of expertise. For instance, MAP-Elites is often used with the same hexapod robot, but with two different sets of definitions: 1) the leg duty-factor for the BD, defined as the time proportion each leg is in contact with the ground, with the average speed for the fitness [10, 11] or 2) the final location of the robot after walking during three seconds for the BD, with an orientation error for the fitness [7, 12, 13]. The resulting behavioural repertoires will find different types of applications. The first set of definitions is designed for fast damage recovery as it provides a diversity of ways to achieve high-speed gaits, while the second one is designed for navigation tasks as the repertoire enables the robot to move in various directions.

3.2 Behaviour Descriptor Definition

To avoid the expertise requirements in the behaviour descriptor definition, several methods have been proposed to enable the automatic definition of the behavioural descriptors. As described previously, AURORA aims to tackle this problem by using a dimensionality reduction algorithm (a Deep Auto-Encoder [31]) to project the features of the solutions into a latent space and use this latent space as a behavioural descriptor space. Other approaches follow similar objectives; for instance, Multi-Container AURORA [5] extends AURORA by considering different complementary BD spaces, all learnt in a simultaneous manner. TAXONS [38] is another QD algorithm adopting the same principle: it demonstrated that this approach can scale to camera images to produce large behavioural repertoires for different types of robots. The same concept was also used in the context of Novelty Search prior to AURORA and TAXONS to generate assets for video games with the DeLeNoX algorithm [30]. Furthermore, a similar method was used to learn goal spaces in an unsupervised manner in IMGEP-UGL [27, 40]. All these methods aim to maximise the diversity of the produced solutions, without specifically considering a downstream task.

A different direction to automatically define a behavioural descriptor is to use Meta-Learning. The idea is to find the BD definition that maximises the performance of the resulting collection of solutions when used in a downstream task. For instance, Bossen et al. [4] consider the damage recovery capabilities provided by a behavioural repertoire as a Meta-Learning objective and search for the linear combination of a pre-defined set of behaviour descriptors that will maximise this objective. A related approach proposed by Meyerson et al. [32] uses a set of training tasks to learn what would be the most appropriate BD definition to solve a "test task". It is also possible to bias the search towards specific regions of the BD space. For instance, Surprise Search [17] and BR-NS [41] use a dynamic surprise-based metric that bias the search towards non-predictable behaviours. Furthermore, the algorithm MAP-Elites with sliding boundaries [15] proposes to automatically adjust the size of cells in MAP-Elites to focus on the most explored regions of the BD space. Instead of adapting the cell size, the Interactive MAP-Elites algorithm [2, 3] enables the users to manually pick the BD dimensions that are the most interesting for their application, before continuing the execution of the MAP-Elites algorithm. Finally, the algorithms SERENE [36] and STAX [37] have proposed to use the concept of emitters, introduced in CMA-MAP-Elites [16], to change the exploration strategy in different regions of the BD space to tackle environments with sparse rewards.

Interestingly, all the methods described above are either strictly task-agnostic or require the definition of a meta-fitness, or a metric to quantify how well the learnt set of behaviours contribute to solving the downstream task. While these works illustrate that it is possible to do this in certain scenarios, we follow the original motivation behind AURORA, which is to discover a set of relevant behaviours with a minimal amount of prior knowledge about the task or the robot.

4 RELEVANCE-GUIDED UNSUPERVISED DISCOVERY OF SKILLS (RUDA)

In this paper, we introduce RUDA, an extension of AURORA with an explicit mechanism to guide the search of solutions towards the most relevant areas of the BD space given a task to solve. That extension consists of three components: (1) A Task Solver which returns a set of relevant individuals; (2) a buffer $B$ of relevant solutions, used to save the most relevant solutions at each step; and (3) an updater of the container metric, that distorts the BD space to promote solutions that are near the ones present in the buffer.

---

**Algorithm 1** RUDA (number of iterations $I$; encoder $E$; target container size $N^\text{target}_C$; container update period $T_C$; downstream task execution period $T_{t \text{ask}}$)

```
1: metric(·,·) ← Euclidean distance
2: $C \leftarrow \emptyset$; $B \leftarrow \emptyset$
3: Initialise $d_{\text{min}}$
4: for $\text{iter} = 1 \rightarrow I$ do
5:   $C \leftarrow \text{QDIteration}(\text{iter}, C, E, \text{metric}(·,·))$
6:   if encoder update expected at iteration $\text{iter}$ then
7:     $\{E, C\} \leftarrow \text{Encoder_Update}(E, C)$
8:     $\{C, d_{\text{min}}\} \leftarrow \text{Manage_Container_Size}(C, d_{\text{min}}, N^\text{target}_C, T_C)$
9:   if $\text{iter}$ multiple of $T_{t \text{ask}}$ then
10:      $\text{pop}_\text{useful} \leftarrow \text{run_task}(C)$
11:      Add $\text{pop}_\text{useful}$ to circular buffer $B$
12:      metric(·,·) ← $\text{update_metrics}(B)$
13: return container $C$
```
4.1 Choice of Relevant Individuals

The choice of relevant individuals indicates which parts of the learned BD space are more useful to the task. We consider a Task Solver that takes as input the current container of individuals from AURORA, and that returns the list of individuals that it used to solve the task. That Task Solver can take several forms. For example, if the task consists of moving to a goal while minimising the number of steps to reach it, then the Task Solver may be a path planner, and the returned individuals are the controllers used before reaching the goal. Similarly, if the task consists of moving forward as fast as possible, the Task Solver may return all the individuals of the container reaching the furthest x position at the end of their episode.

4.2 Circular Buffer of Relevant Individuals

The individuals returned by the Task Solver are then stored in a circular buffer $\mathcal{B}$. This way, the buffer keeps track of the individuals that are relevant for the downstream task. In all our experiments, that buffer was chosen to be circular to prevent “outdated” individuals from staying in the buffer. Indeed, between two calls to the Task Solver, the content of the container may change: it may contain as possible, the Task Solver may return all the individuals of the buffer was chosen to be circular to prevent “outdated” individuals from staying in the buffer. Indeed, between two calls to the Task Solver, the content of the container may change: it may contain all the individuals that are more suitable for solving the task. In that case, if some individuals have not been added to the buffer for a long time, this means there are more suitable individuals present in the container.

4.3 Update Relevance-based Distance Metric

The role of the relevance-based distance metric is to make it easier for new individuals to be added to the container if they are closer to the relevant individuals present in $\mathcal{B}$. When a new individual attempts to be added to the container, a relevance score is assigned to it. That relevance score estimates the proximity of this individual to the circular buffer of relevant individuals $\mathcal{B}$; the closer an individual is to $\mathcal{B}$, the higher is its relevance score. That relevance score is calculated as the inverse mean Euclidean distance in the BD space to the $k$-nearest neighbours from the buffer $\mathcal{B}$:

$$relev_{ind} = \left(\frac{1}{k} \sum_{b \in \text{NN}(\mathcal{B}, b)} \| b - b' \|_2 \right)^{-1}$$  \hspace{1cm} (1)

Then this relevance score is used to define a new distance metric, that distorts the BD space in favour of individuals having a high relevance score:

$$\forall x, \text{metric}(\text{indiv}, x) = relev_{ind} \| x^{\text{indiv}} - b' \|_2$$  \hspace{1cm} (2)

That new metric is used to estimate all the distances between individuals in the container; it is also taken into account when calculating novelty scores. Thus, if an individual is close to $\mathcal{B}$, then its relevance score will be high, and all the distances seen from the point of view of this individual will be higher. This means that individuals with higher relevance scores have higher chances of being added to the container, and the container will present higher densities of individuals in the regions near the buffer $\mathcal{B}$.

5 EXPERIMENTAL SETUP

5.1 Agent: Neural-network controlled hexapod

In all our experiments, our agent consists of a simulated hexapod robot controlled via a neural network controller. The hexapod robot presents 18 controllable joints (3 per leg). Each leg $l$ has two Degrees of Freedom: its hip angle $\alpha_l$ and its knee angle $\beta_l$. The ankle angle is always set to the opposite of the knee angle.

The neural network controller is a Multi-Layer Perceptron with a single hidden layer of size 8. It takes as input the current joint angles and velocities $(\alpha_l, \beta_l, \dot{\alpha}_l, \dot{\beta}_l)_{1 \leq l \leq 6}$ (of dimension 24), and outputs the target values to reach for the joint angles $(\alpha_l, \beta_l)_{1 \leq l \leq 6}$ (of dimension 12). The controller has in total $(24+1) \times 8 + (8+1) \times 12 = 308$ independent parameters, operates at a frequency of 50Hz, while each episode lasts for 3 seconds. The environment and the controller are deterministic.

5.2 Dimensionality Reduction Algorithm of AURORA and RUDA

For each evaluated individual, the collected sensory data corresponds to the joint positions, and torso positions and orientations collected at a frequency of 10Hz. In the end, this represents 18 streams $((\alpha_l, \beta_l)_{1 \leq l \leq 6}$, and the positional and rotational coordinates of the torso), with each of those streams containing 30 measurements. The dimensionality reduction algorithm used in AURORA and RUDA is a reconstruction-based auto-encoder. The input
data is using two 1D convolutional layers, with 128 filters and a kernel of size 3. Those convolutional layers are followed by one fully-connected linear layer of size 256. The decoder is made of a fully-connected linear layer of size 256, followed by three 1D deconvolutions with respectively 128, 128 and 18 filters.

The loss function used to train the encoder corresponds to the mean squared error between the sensory data passed as input and the reconstruction obtained as output. The training is performed using the Adam gradient-descent optimiser [25] with \( \beta_1 = 0.9 \) and \( \beta_2 = 0.999 \). As the encoder needs to be updated less and less frequently as the number of iterations increases, the interval between two encoder updates linearly increases over time (as in previous work [18]): encoder updates occur at iterations 0, 10, 30, 60, 100...

5.3 **Downstream Tasks and QD Tasks**

We make the distinction between downstream tasks and QD tasks. A QD task evaluates the diversity and the performance of individuals present in the container returned by a QD algorithm. A downstream task corresponds to a high-level problem that we intend to solve using the QD container.

For each experiment, we define a **Task Solver**, a **Task Score**, a **Container Score** and an **associated QD Task**. The Task Solver chooses individuals from the container that are relevant to solve the task, and sends them to the buffer \( B \). The Task Score characterises the overall performance of the Task Solver in solving the downstream task; contrary to the fitness score which evaluates the performance of an individual, the task score indicates the performance of a full container. The Container Score is a theoretic measure of the expected performance, taking into account all individuals present in the container. Finally, the associated QD Task is a set of BD and expected performance, taking into account all individuals present in the container returned by a QD algorithm. A downstream task corresponds to the **duty factor** as hand-coded for succeeding at the downstream task; consequently, the QD Task is defined to be strongly correlated with the downstream task.

We consider three downstream tasks with the hexapod agent: **Navigation**, **Moving Forward**, and **Half-roll**.

5.3.1 **Navigation**. The Navigation task is inspired from (Chatzilygeroudis et al.) [7] and (Kaushik et al.) [23]. The container provided by the QD algorithm is used as a basis for choosing actions to reach successive goals, characterised by their \( x,y \) position.

**Task Solver**: A goal is randomly positioned goal at a distance of approximately 5 meters from the starting position of the robot. The solver then iteratively chooses the individual from the repertoire that gets the closest to the goal, runs it for three seconds, and repeat the previous steps until the distance to the goal is lower than 5cm.

**Task Score**: We compute the number of actions required to reach a fixed number of 50 goals at specific positions, all separated by approximately 7 meters. The task score then corresponds to the negated average number of actions required to reach each separate goal.

**Container Score**: It corresponds to the coverage of the container in \( x,y \) space. That coverage is calculated by discretising the space of possible final positions \([−2.2 m, 2.2 m] \times 50 \times 50 \) grid. The coverage then corresponds to the number of grid cells containing at least one individual from the unstructured container.

**Associated QD Task**: This task aims at finding a container of controllers reaching diverse final \((x_T, y_T)\) positions in \( T = 3 \) seconds following circular trajectories. Hence the hand-coded BD in this case is \((x_T, y_T)\), while the fitness is the final orientation error as defined in the literature [7, 12].

5.3.2 **Moving Forward**. This task is inspired from the work of Cully et al. [10], which aims to obtain a container with a variety of ways to move forward at a high velocity.

**Task Solver**: it takes as input the entire container obtained from the AURORA phase, and returns the 10 individuals reaching the furthest position along the \( x \) axis after 3 seconds. Those individuals are added to the buffer, with replacement.

**Task Score**: The score associated to that task corresponds to the maximal position (along the \( x \) axis) achieved by the container.

**Container Score**: corresponds to the mean position (along the \( x \) axis) of all individuals in the container.

**Associated QD Task**: The associated QD task follows the definition from Cully et al. [10], with the Duty Factor as hand-coded BD. The Duty Factor evaluates the proportion of time the each leg is in contact with the ground. The QD fitness promotes individuals achieving the highest \( x \) position at the end of the episode.

5.3.3 **Half-roll**. The Half-roll task aims at obtaining a variety of ways to reach a position where the pitch angle of the robot is equal to \(-\pi/2\) (i.e. where the robot falls on its back).

**Task Solver**: In this downstream task, the task solver returns the 10 individuals whose pitch angle is the closest to \(-\pi/2\).

**Task Score**: The score associated to that task is the negated minimal distance to a pitch angle of \(-\pi/2\).

**Container Score**: The container score corresponds to the negated average distance to a pitch angle of \(-\pi/2\).

**Associated QD Task**: The QD task associated to the half-roll downstream task aims at finding a diversity of ways to perform half-rolls, i.e. behaviours such that the hexapod ends with its back on the floor. In particular, the behaviours are characterised via the final yaw and roll angle of the torso. And the fitness is the distance of the pitch angle \( \alpha_{pitch} \) of the hexapod to \(-\pi/2\).

\[
\begin{align*}
\delta_{indiv} &= \left( \alpha_{yaw, indiv}, \alpha_{roll, indiv} \right) \\
\mathcal{f}_{indiv} &= -\left| \alpha_{pitch} - \left( -\frac{\pi}{2} \right) \right|
\end{align*}
\]

5.4 **Compared algorithms and variants**

We study two categories of variants: those that learn their BD autonomously, and those based on hand-defined BDs. Those variants are summarised in Table 3.

5.4.1 **Algorithms with Unsupervised Behavioural Descriptors**.

| Variant       | BD                        | Uses Downstream Task Solver | Has Fitness |
|---------------|---------------------------|-------------------------------|-------------|
| RUDA          | Unsupervised              | ✓                             | ✗           |
| AURORA        | Unsupervised              | ✗                             | ✓           |
| R-MeS         | Mean streams              | ✓                             | ✗           |
| MeS           | Mean streams              | ✓                             | ✗           |
| HC            | QD Task BD                | ✗                             | ✓           |

Table 3: Characteristics of the variants of RUDA under study, grouped based on their type of Behavioural Descriptor (BD).
In addition to RUDA, we also study a version of RUDA that does not have a relevance-based mechanism, which corresponds to AURORA.

**RUDA**: The dimensionality of the learnt BD space is set to 10. Similarly to prior work [18, 38], we suppose RUDA is completely QD-task-agnostic, which means that not only it automatically learns its BDs, but it also does not have access to the fitness function of the QD task \( f_{\text{indio}} \).

**AURORA**: It corresponds to RUDA without the relevance-based mechanism that updates the container metric. Contrary to RUDA, AURORA does not interact with a downstream task, but we consider AURORA has access to the fitness function \( f_{\text{indio}} \) of the QD task. This way, we will be able to evaluate the difference between using a fitness function, and using a relevance-based mechanism.

### 5.4.2 Variants with Hand-defined Behavioural Descriptors.

We consider three different variants having hand-defined BDs.

**Hand-Coded (HC)**: considers a low-dimensional BD that is defined by hand as the Behavioural Descriptor of the QD task (e.g., see Table 2). The HC variant corresponds to a standard QD algorithm with an unstructured archive [11] and the mechanism of container size control introduced in previous work [18].

**R-MeS**: considers a Behavioural Descriptor with more information from the collected sensory than the Hand-Coded variant explained above. To calculate its BD, this variant considers the 18 data streams collected by the hexapod, and average each one of them to obtain a BD of dimension 18. This variant uses the same relevance-based bias as RUDA; and similarly to RUDA, it does not have access to the fitness function of the QD task \( f_{\text{indio}} \). The comparison between this variant and RUDA will be useful to evaluate the usefulness of the automatic dimensionality reduction algorithm.

**MeS**: presents the same characteristics as R-MeS, except that it does not use any relevance-based mechanism. As AURORA, MeS does not have access to the downstream task, but it uses the fitness function \( f_{\text{indio}} \).

### 5.5 Implementation Details

All our experiments were run for 15,000 iterations with a uniform QD selector. We only use polynomial mutations as variation operators with \( \eta_m = 10 \), and a mutation rate of 0.3. The target container size \( N_{\text{target}} \) of all algorithms is set to 5,000 for the Moving Forward and the Half-roll tasks. In the case of the Navigation Task, we set it to 1,500 to obtain appropriate comparisons between the different approaches. To keep the container size around \( N_{\text{target}} \), we perform a container update every \( T_C = 10 \) iterations (as explained in Section 2.1). For relevance-based algorithms (RUDA and R-MeS), the downstream task execution period \( T_{\text{task}} \) is set to 10 iterations, the maximal buffer size is set to 200, and relevance scores (see Eq. 1) are calculated with \( k = 15 \) nearest neighbours.

All our implementation is based on Sferes2 [34] and uses the DART simulator [28], while the auto-encoders are coded and trained using the C++ API of PyTorch [39].

For each downstream task, each variant was run for 10 replications, and the statistical significance of the comparisons is measured using the Wilcoxon rank-sum test, with a Holm-Bonferroni correction [19]. To facilitate the replication of the results, we stored...
We first aim at demonstrating that RUDA finds containers of individuals that are useful with respect to the task to solve. The first results, this results in higher container scores for relevance-based mechanisms compared to AURORA and MeS (Fig. 3-bottom, \( p < 2 \times 10^{-3} \)).

In particular, in the Moving Forward task, RUDA and R-MeS perform better than all other variants (\( p < 2 \times 10^{-3} \)). That is likely due to the shifting of the container distribution towards the most relevant individuals. Indeed, as most individuals in the container have a high \( x \) position, then there is a high probability to generate individuals who also have a high \( x \) position, and there are more chances to generate better offspring. In other words, the QD selection is biased in the direction of the most useful individuals, which improves the results obtained by the relevance-based variants.

Also, in the Moving Forward and Half-roll tasks, RUDA also leads to higher container scores compared to the other relevance-based variant, R-MeS, (Fig. 3-bottom, \( p < 5 \times 10^{-3} \)). This shows that having a flexible encoding mechanism is useful for adapting the BD to maximise the relevance of the archive. Moreover, in the tasks where the relevance is directly correlated with the fitness (Moving Forward and Half-roll), RUDA manages to find encodings that favour the most relevant individuals (see Fig. 2).

In the case of the Navigation task, we can see that RUDA achieves significantly better task scores than AURORA (Fig. 3, \( p < 2 \times 10^{-3} \)). One possible explanation for this is that the containers obtained by RUDA present a wider coverage in terms of \( x, y \) final positions than AURORA (Fig. 3-bottom, and 2). That may also explain why using Hand-Coded BDs leads to the best scores. We can also note that the coverage follows the peculiar distributions of the relevant individuals in the buffer: there is a higher-density of individuals executing either small manoeuvres or very large displacement. This makes sense as an effective way to solve this downstream task is to first walk as fast as possible in the direction of the goal and then adjust during the final approach. Hence, RUDA takes in average fewer actions to reach each goal, leading to a better task score.

### 6.2 Container Diversity

As explained before, the purpose of RUDA is not only to find specialised containers, but also containers that are diverse. This means that we expect RUDA to return a container of diverse relevant behaviours. However, the notion of relevance is task-dependent, so we need to adjust the analysis to the task under study. In particular, we take the containers returned by the different algorithms, and we project them in hand-coded BD spaces that are unrelated to the task. Then we evaluate the achieved coverage in the BD space by considering the coverage for several minimum fitness scores.

#### 6.2.1 Coverage per minimum fitness

The containers returned by QD algorithms always result in a trade-off between the diversity of solutions and the total quality of the entire container. To evaluate the coverage depending on the quality of the solutions, we study the evolution of the coverage given several minimum fitness scores. With a fitness \( f_{\min} \) and a BD space discretised into a grid, the "coverage given minimum fitness \( f_{\min} \)" is defined as: the percentage of cells with individuals whose fitness is higher than \( f_{\min} \) [14].

#### 6.2.2 Navigation Task

In the Navigation task, we have seen that RUDA and the Hand-Coded baseline return containers with diverse \( x, y \) positions (Figs. 2 and 3). We intend to show that RUDA also discovers diverse ways to move to those positions. To evaluate this gait diversity, we project the containers in the BD space “Duty Factors”, and evaluate the coverage in that BD space. Even when considering all individuals \( f_{\min} = -3.5 \), the container returned by RUDA on the Navigation task exhibits a higher coverage than the hand-coded container (Figs. 4 and 6, all \( p \)-values are \( < 10^{-2} \)). The hand-coded variant does not have any incentive to promote diversity in the BD space of Duty Factors as it only attempts to maximise the diversity of final positions. This may explain the poor performance of the Hand-coded variant in terms of diversity.

It is also worth noting that AURORA and MeS achieve a higher diversity than the relevance-based variants. That discrepancy is likely due to the fact that AURORA and MeS do not intend to specialise for any task. On the contrary, as RUDA returns a container specialised for a downstream task, some diversity is inherently lost.

Finally, we see that the containers returned by the non-relevance based algorithms contain more high-performing individuals than RUDA and R-MeS. This phenomenon is due to the fact that RUDA does not optimise for that fitness function, which promotes circular trajectories. Instead, here RUDA optimises for the downstream Navigation Task, and it is possible to succeed at that task while using individuals which are not performing circular trajectories.

#### 6.2.3 Moving Forward Task

While RUDA produces containers of controllers to move forward, we also expect those containers to exhibit diversity in terms of other features. We estimate this additional diversity by projecting the obtained containers in two classical BD...
Figure 4: Visualisations of the obtained containers from the “Navigation” task, for all algorithms under study. The containers are presented after having projected the individuals on the Duty Factor BD space. Each coloured pixel represents a grid cell of the BD space that has been filled with at least one individual. The colour is representative of the fitness. Note that the grids presented have six dimensions, they are presented using the same technique as in the work of Cully et al. [10].

Figure 5: Visualisations of the obtained containers from the “Moving Forward” task, for all algorithms under study. The first row presents the containers after having projected the individuals on the Duty Factor BD space. The second row shows the obtained containers after a projection on the Body Orientation BD space.

Figure 6: Coverage per minimum fitness, considered with different downstream tasks and projected in different BDs.

spaces used for the Hexapod uni-directional task [10]: the Duty Factor, and the Body Orientation. The Duty Factor BD space is the same as defined previously, it is the one used by the hand-coded variant on the Moving Forward task. The Body Orientation BD space characterises the amount of time spent by the torso of the hexapod at given orientations (more details can be found in the work of Cully et al. [10]).

Figures 5 and 6 show that the containers of RUDA present a lower coverage compared to the other variants that are not relevance-based ($p < 10^{-2}$). However, when considering a minimum fitness score above 1.4, which corresponds to a minimal $x_I$ of 1.4 meters, the diversity exhibited by RUDA is better ($p < 2 \times 10^{-3}$). This confirms the fact that the container returned by RUDA not only specialises for a downstream task, but also still exhibits diversity in its specialisation.

7 CONCLUSION AND FUTURE WORK

In this paper, we introduced RUDA, an extension of AURORA which aims to autonomously generate behavioural repertoires with a higher density of behaviours in the regions of the BD space that are relevant for the considered tasks. Our experimental evaluation demonstrated that RUDA can adjust the distribution of behaviours depending on the situation, over the three considered tasks.

We also noticed that specialising for a downstream task often reduces the behavioural diversity, or moves the diversity towards the most relevant areas of the behavioural spaces. It would be interesting to study whether all future methods to distort the learnt BD space will show the same influence on the trade-off between specialisation and diversity. It would also be relevant to try our approach on more complex problems, such as multi-tasks problems [35].

ACKNOWLEDGMENTS

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) grant EP/V006673/1 project REcoVER.
