



**HAL**  
open science

# Discovering and Exploiting Sparse Rewards in a Learned Behavior Space

Giuseppe Paolo, Miranda Coninx, Alban Laflaquière, Stéphane Doncieux

► **To cite this version:**

Giuseppe Paolo, Miranda Coninx, Alban Laflaquière, Stéphane Doncieux. Discovering and Exploiting Sparse Rewards in a Learned Behavior Space. *Evolutionary Computation*, inPress, pp.1-28. hal-04239313

**HAL Id: hal-04239313**

**<https://hal.science/hal-04239313>**

Submitted on 12 Oct 2023

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

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

---

# Discovering and Exploiting Sparse Rewards in a Learned Behavior Space

**Giuseppe Paolo**

giuseppe.paolo@softbankrobotics.com

AI Lab, SoftBank Robotics Europe

Sorbonne Université, CNRS, Institut des Systèmes Intelligents et de Robotique, ISIR

Paris, France

**Miranda Coninx**

miranda.coninx@sorbonne-universite.fr

Sorbonne Université, CNRS, Institut des Systèmes Intelligents et de Robotique, ISIR

Paris, France

**Alban Laflaquière**

alaflaquiere@softbankrobotics.com

AI Lab, SoftBank Robotics Europe

Paris, France

**Stephane Doncieux**

stephane.doncieux@sorbonne-universite.fr

Sorbonne Université, CNRS, Institut des Systèmes Intelligents et de Robotique, ISIR

Paris, France

---

## Abstract

Learning optimal policies in sparse rewards settings is difficult as the learning agent has little to no feedback on the quality of its actions. In these situations, a good strategy is to focus on exploration, hopefully leading to the discovery of a reward signal to improve on. A learning algorithm capable of dealing with this kind of settings has to be able to (1) explore possible agent behaviors and (2) exploit any possible discovered reward. Exploration algorithms have been proposed that require the definition of a low-dimension behavior space, in which the behavior generated by the agent's policy can be represented. The need to design a priori this space such that it is worth exploring is a major limitation of these algorithms. In this work, we introduce STAX, an algorithm designed to learn a behavior space on-the-fly and to explore it while optimizing any reward discovered. It does so by separating the exploration and learning of the behavior space from the exploitation of the reward through an alternating two-step process. In the first step, STAX builds a repertoire of diverse policies while learning a low-dimensional representation of the high-dimensional observations generated during the policies evaluation. In the exploitation step, emitters optimize the performance of the discovered rewarding solutions. Experiments conducted on three different sparse reward environments show that STAX performs comparably to existing baselines while requiring much less prior information about the task as it autonomously builds the behavior space it explores.

## Keywords

Sparse Rewards, Novelty Search, Emitters, Evolutionary Algorithms, Quality Diversity

## 1 Introduction

For an embodied agent whose goal is to learn a policy capable of solving a task, situations of *sparse rewards* can be difficult to deal with. The reason is that many policy-learning algorithms work by optimizing a *reward function* providing feedback on the performances of the policy. A well-designed reward function has to provide a reward often enough so the agent can know how good each performed action is (Sutton and Barto, 2018). These kinds of rewards are called

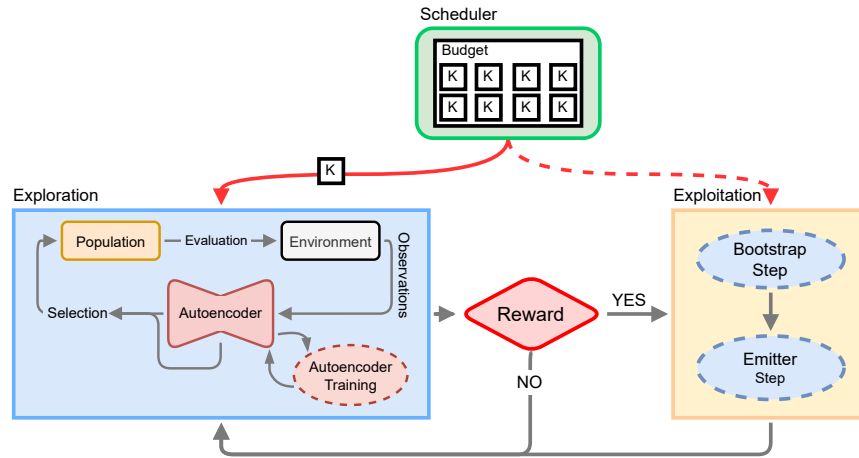


Figure 1: STAX consists of an exploration and an exploitation process alternating thanks to a scheduler. During exploration, the algorithm explores and learns a representation of the behavior space through an AE trained online. Any discovered reward is then exploited in the exploitation step through emitters.

*dense rewards*. On the contrary, in sparse rewards settings, this feedback is provided sparingly, only after a given amount of time is passed or if a specific situation happens. In these situations, it is difficult for a learning agent to evaluate how good a policy is and how appropriate each action is to each situation. This can reduce the performance or even hinder the learning of a good policy. An example of this can be a robotic arm learning how to pick an object. The simplest way of rewarding the agent is to give the reward when the arm picks the object, while designing a reward that could lead the arm to pick the object is very hard. For these reasons, when reward feedback is not readily available, a good strategy is to focus on *exploration*, with the goal of finding a reward in the future.

Following this strategy, the way exploration is performed becomes fundamental. Standard Reinforcement Learning (RL) algorithms, as described by Sutton and Barto (2018), perform exploration through random actions, a strategy that renders unlikely to find rewards if they are sparse enough. This problem has been addressed with the introduction of different approaches, based on both RL methods, Evolutionary Algorithms (EAs) or a mix of both (Sigaud, 2022). Among them, Novelty Search (NS) is an EA that completely discards any performance information, focusing solely on exploration by looking for a set of policies whose behaviors are as different as possible (Lehman and Stanley, 2008). This is done in a hand-designed space, the behavior space ( $\mathcal{B}$ ), in which the behavior of each one of the generated policies is represented in order to evaluate their diversity. The development of NS has led to the birth of the evolution-based *divergent search* family of algorithms, also known as Quality-Diversity (QD) (Pugh et al., 2016; Cully and Demiris, 2017). These methods, in addition to focusing on pure exploration through divergent search, can also optimize the performances of the discovered policies. This grants a strong advantage over methods like NS that tend to produce low-performing solutions with respect to the a posteriori evaluation on a rewarding task. Nonetheless, the exploration abilities of these approaches, NS included, are often limited by the need to hand-design  $\mathcal{B}$ . While this allows the designer to define what aspects of the problem need to be explored, it also increases the engineering cost of these methods while limiting the range of problems to which they can be applied. To address this issue, researchers have introduced methods that can autonomously learn  $\mathcal{B}$  through *representation learning approaches*, thus reducing the amount of prior information needed for the design of the representation itself (Liapis et al., 2013;

Paolo et al., 2020; Grillotti and Cully, 2022). Supporting this approach, Hagg et al. (2020) have shown that autonomously learning  $\mathcal{B}$  generates higher diversity of solutions compared to hand-designed  $\mathcal{B}$ . Notwithstanding the good results obtained by these methods, they are still limited either by the discarding of reward-related information of NS (Liapis et al., 2013; Paolo et al., 2020) or by the need to discretize the learned space (Grillotti and Cully, 2022).

In this paper, we introduce the STAX algorithm, a method that can perform exploration in a search space that is autonomously learned at execution time, while also optimizing any possible discovered reward. As with NS, this exploration is completely reward-agnostic, but contrary to this method, once an area of the search space is discovered to contain a reward, STAX performs a local search in this area with the goal to optimize the total obtained reward. This optimization is performed through *emitters*, a concept introduced by Fontaine et al. (2020), consisting of instances of reward-based EAs used to perform local search in an area of the whole  $\mathcal{B}$ . The idea of emitters was used in Sparse Reward Exploration via Novelty search and Emitters (SERENE) (Paolo et al., 2021) to optimize any reward discovered during the search performed by NS. At the same time, SERENE still relies on a hand-designed behavior space in which to perform the search. STAX builds on SERENE by removing this requirement through the use of an autoencoder (AE) to learn the behavior space online while performing the search (Grillotti and Cully (2022); Paolo et al. (2020)).

SERENE augmented TAXONS (STAX) deals with sparse reward problems by separating the exploration and the learning of the unknown search space from the exploitation of any possible reward through an alternating two-step process. In the first step, the algorithm explores the search space guided by the low-dimensional representation of the policies behavior given by the AE. At the same time, this representation is learned by training the AE on the data collected during the evaluation of the discovered policies. When rewards are found, they are exploited in the second step through the use of *emitters*, in a way similar to SERENE (Paolo et al., 2021). The clear separation between exploration and exploitation has many advantages. The two processes often push the optimization in different directions: exploration requires trying as many things as possible, while exploitation requires getting better at the things we already know. Working on them separately, then allows doing both without degrading performances, as it can happen in multi-objective approaches like NSGA-II (Paolo et al., 2021). Moreover, the decoupling of exploration from exploitation enables using different strategies for either of the two processes in a more modular approach.

To recap, STAX performs three main tasks: (1) learning a behavior space while (2) exploring it, and (3) efficiently exploiting a reward once it is found. The method builds on NS by adding an AE to learn a low dimensional representation of the search space. Moreover, the reward is exploited through emitters to quickly improve on rewards. The advantages provided by STAX are twofold: (1) it can deal with sparse rewards situations by discovering and quickly optimizing the rewards, thanks to the separation of the exploration process from the exploitation of the reward provided by the use of emitters; (2) by autonomously learning the  $\mathcal{B}$ , it removes the limitation of classical divergent-search approaches requiring a hand-designed space, thus reducing the amount of prior information needed at design time. All of this allows STAX to deal with sparse reward environments with minimum prior information required about the task at design time.

The paper is organized as follows: Sec. 2 will present an overview of related work and the methods on which STAX builds. The STAX method itself is detailed in Sec. 3, while the experimental settings on which it has been tested are shown in Sec. 4. The obtained results are shown and discussed in Sec. 5. The paper concludes with Sec. 6, in which possible extensions and improvements are discussed.

## 2 Background and related work

This section presents an overview of other works on the sparse rewards problem, together with an explanation of how NS and *emitters* work.

### 2.1 Sparse Reward

For many policy learning approaches, the reward function is fundamental: it is through this function that the designer communicates to the learning agent what is the goal the policy should solve (Sutton and Barto, 2018). If the reward signal is given sparingly, after a lot of time, or only if certain conditions are met, the agent can often find itself in situations in which no reward is present, thus with no signal to drive the learning. To address this issue, many approaches have been proposed. Some of these approaches rely on *reward shaping* (Mataric, 1994; Ng et al., 1999), a technique consisting of augmenting the original reward function with additional features that are supposed to provide the agent with better guidance in solving the task (Hu et al., 2020; Berner et al., 2019; Trott et al., 2019). Another successful strategy is the self-assigning of goals. This can be done either by using information from previously encountered situations (Andrychowicz et al., 2017), or by using the representations of an unsupervised learning algorithm over a distribution of possible targets (Nair et al., 2018).

A different approach is based on Intrinsic Motivation (IM) (Oudeyer and Kaplan, 2009; Aubret et al., 2019), by having the agent generate its own learning signal, without the need for any environmental reward. This can be obtained by estimating the novelty of a state by considering how often that state has been visited (Bellemare et al., 2016; Burda et al., 2018). The less novel a state is, the more the agent is pushed to go elsewhere, thus performing more exploration. Goal-Exploration Processes (GEP) are another family of algorithms that use the self-assignment of goals to foster exploration (Baranes and Oudeyer, 2013; Forestier et al., 2022; Laversanne-Finot et al., 2018). Forestier et al. (2022) use this to first learn a goal-parametrized policy and then use this policy to solve the given task. These approaches have also been used with two-phase strategies to help separate the exploration process from the exploitation of the possible discovered rewards Colas et al. (2018); Ecoffet et al. (2019).

*Divergent-search algorithms* are a family of EAs specifically designed to focus on exploration, rendering them naturally suited to deal with sparse reward situations (Lehman and Stanley, 2008; Cully and Demiris, 2017; Pugh et al., 2016). The first introduced method of this family is NS, introduced by Lehman and Stanley (2008), which works by completely ignoring any reward signal in order to generate a set of solutions as diverse as possible. Inspired by NS, many other methods have been introduced that not only focus on the diversity of the solutions but also optimize their performances with respect to a given objective. This gave rise to a new family of methods called QD (Cully and Demiris, 2017; Pugh et al., 2016; Cully et al., 2015; Eysenbach et al., 2018; Lehman and Stanley, 2011; Paolo et al., 2021; Mouret and Clune, 2015). Moreover, given the great exploration abilities provided by divergent-search algorithms, some researchers combined them with RL methods to better deal with sparse reward situations (Colas et al., 2018; Cideron et al., 2020).

### 2.2 Novelty Search

NS is an EA that drives the search by focusing on maximizing the diversity of a set of solutions (Lehman and Stanley, 2008). To do this, the algorithm uses a metric called *novelty*, calculated in an *hand-designed behavior space*  $\mathcal{B}$  in which the behavior of each policy is represented. This space, in the literature also called *outcome space* (Paolo, 2020), is at the heart of NS and needs to be tailored to the problem at hand by using prior knowledge of the system and the task.

The algorithm works by evaluating each policy, parametrized by a set of parameters  $\theta_i \in \Theta$ , on the system for  $T$  time-steps. During this evaluation, the system traverses a set of states  $s_t$

generating a trajectory of traversed states  $\tau_s = [s_0, \dots, s_T]$ . These states are observed by the agent through some sensors, generating a corresponding trajectory of observations  $\tau_{\mathcal{O}} = [o_0, \dots, o_T]$ , where  $o_t \in \mathcal{O}$  is the, possibly partial, observation of state  $s_t$ . These observations can be generated in different ways, depending on the setting. If the states are known, the agent can directly work with them, in which case  $o_t = s_t$ . In other situations, the state needs to be observed through sensors, in which case  $o_t$  would be a, possibly partial, representation of  $s_t$ . The trajectory of observations  $\tau_{\mathcal{O}}$  is then used to extract the corresponding behavior descriptor  $b_i \in \mathcal{B}$  of the policy  $\theta_i$  through an observer function  $O_{\mathcal{B}} : \mathcal{O} \rightarrow \mathcal{B}$ . The whole process is summarized by using a *behavior function*  $\phi : \Theta \rightarrow \mathcal{B}$  that directly maps a policy to its behavior descriptor:

$$\phi(\theta_i) = b_i. \quad (1)$$

Once the behavior descriptors of all the policies in a population have been calculated, the novelty of a policy  $\theta_i$  in the population can be obtained by measuring the average distance of its behavior descriptor with respect to the descriptors of its  $k$  closest policies. The higher this distance, the more novel the behavior of a policy is considered. The novelty  $\eta(\theta_i)$  is calculated through the following equation:

$$\eta(\theta_i) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(b_i, b_j) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(\phi(\theta_i), \phi(\theta_j)) \quad (2)$$

where  $J$  is the set of indexes of the  $k$  closest policies to  $\theta_i$  in  $\mathcal{B}$ .

At each generation, the novelty of the policies is calculated and the ones with the highest novelty are selected to be part of the next generation population. At the same time  $N_Q$  policies are selected at each generation to be stored into an *archive*  $\mathcal{A}_{\text{Nov}}$ . The function of the archive is to keep track of the already explored areas of the search space, pushing the search towards less visited areas. This is done by selecting the  $|J|$  policies used for the novelty calculation in equation 2 not only from the current population but also from the archive.

### 2.3 Learning a behavior descriptor

At the core of many divergent search algorithms lies a hand-designed  $\mathcal{B}$ . The need to hand-design this space poses strong limitations for the application of these methods to various problems in which the factors important for the exploration are not clear. To overcome this problem, many approaches that use representation learning methods to learn a low-dimensional representation of the behavior of the policy have been recently proposed (Paolo et al., 2020; Cully, 2019; Liapis et al., 2013).

Cully (2019) uses the learned low-dimensional representation to describe the behavior of the policy and select in which cell of the MAP-Elites grid the policy itself belongs. At the same time, Task Agnostic eXploration of Outcome spaces through Novelty and Surprise (TAXONS) (Paolo et al., 2020) selects the policies not only based on the novelty calculated through the learned low-dimensional representation but also on the reconstruction error of the AE through a metric called *surprise* (Gaier et al., 2019). The idea behind this is that the higher the reconstruction error, the less often a behavior has been seen, thus the more novel it is. This is similar to the approaches introduced by Burda et al. (2018) and Salehi et al. (2021). STAX uses TAXONS to learn the low-dimensional representation of the behavior of a policy during the exploration phase, thus removing the need to hand-design  $\mathcal{B}$ . At the same time, rather than selecting the policies according to only one of the two metrics, novelty or surprise, as done by TAXONS, it uses the NSGA-II Multi-Objective optimization (MOO) approach (Deb et al., 2002) to combine both objectives and select the policies at each generation.

## 2.4 Emitters

Among QD algorithms worth of notice are approaches using *emitters*. Introduced by Fontaine et al. (2020) and later used by Cully (2021) and Paolo et al. (2021), emitters are instances of local-search reward-based EAs instantiated during the global search performed by another EA, allowing the quick exploration of a small area of the search space while optimizing on the reward. There is no limitation on the kind of algorithm to use as an emitter. In the work from Fontaine et al. (2020), the CMA-ME algorithm uses MAP-Elites in conjunction with estimation-of-distribution emitters. The algorithm works by sampling a policy  $\theta_i$  from the MAP-Elites archive and using it to initialize the population of an emitter  $\mathcal{E}_i$ , which is then evaluated until a termination condition is met. The policies discovered are added to the MAP-Elites archive according to a given addition strategy. Once an emitter is terminated, another policy  $\theta_j$  is selected from the MAP-Elites archive to initialize another emitter. The cycle is repeated until the whole evaluation budget is depleted.

Another method using emitters is SERENE. Introduced by Paolo et al. (2021), the algorithm is based on NS and targets explicitly sparse rewards problems. Contrary to CMA-ME, SERENE works through an alternating two-stage process, one performing exploration, the other exploiting the found rewards. Exploration is done through NS over the hand-designed  $\mathcal{B}$ . Once a reward is discovered, it is exploited in the exploitation step when emitters are launched over the rewarding area of the search space  $\mathcal{B}_R \subseteq \mathcal{B}$ . This two-steps process allows the algorithm to easily deal with sparse rewards settings in which, while the search can be global, the optimization of the reward has to be local around the rewarding policy.

The method introduced in this work augments SERENE with the ability to autonomously learn  $\mathcal{B}$  through a strategy similar to TAXONS (Paolo et al., 2020). In the next sections we will detail how STAX works and how, by taking advantage of emitters and the unsupervised learning of the behavior space, it is possible to quickly explore an unknown search space while efficiently optimizing any possible discovered reward.

## 3 Method

STAX deals with *sparse rewards* settings by separating the search process into two alternating sub-processes: one performing *exploration* of the search space and another performing *exploitation* of any discovered reward. The algorithm starts with the *exploration* phase and then the two processes are alternated through a *meta-scheduler*. The task of the meta-scheduler is to split the total evaluation budget  $Bud$  in small chunks of size  $K_{Bud}$  and assign them to either one of the two sub-processes. In the *exploration* phase, STAX learns a  $\mathcal{B}$  from high dimensional observations of the environment through an AE, and explores it. Meanwhile, during the *exploitation* phase, the discovered rewarding policies are optimized through *emitters*, instances of local-search reward-based optimization algorithms. Note that, while in this work we use an *elitist EA*, any kind of optimization algorithm can be used as emitter. Once the whole evaluation budget  $Bud$  is depleted, the algorithm returns two collection of policies: the *novelty archive*  $\mathcal{A}_{Nov}$ , containing the diverse-behavior policies found during the exploration phase, and the *reward archive*  $\mathcal{A}_{Rew}$ , containing the reward-optimized policies found during the exploitation phase. The whole process is designed in a way that allows the discovery of different high-reward policies with minimal prior information about the task.

There are two aspects of STAX that are worth highlighting: the autonomous learning and exploration of the behavior space and the optimization of the reward through emitters. Autonomously learning the  $\mathcal{B}$  allows to reduce the amount of prior information needed to solve the task by removing the need to hand-design  $\mathcal{B}$ . This is achieved by learning a low-dimensional representation of this space through an AE, directly from high-dimensional observations collected during the policy evaluation, in a fashion similar to TAXONS (Paolo et al., 2020).

---

**Algorithm 1: STAX**


---

**INPUT:** evaluation budget  $Bud$ , budget chunk size  $K_{Bud}$ , population size  $M$ , emitter population size  $M_{\mathcal{E}}$ , offspring per policy  $m$ , mutation parameter  $\sigma$ , number of policies added to novelty archive  $Q$ , AE training interval  $TI$ , randomly initialized AE, number of bootstrap generations  $\lambda$ ;

**RESULT:** Novelty archive  $\mathcal{A}_{Nov}$ , rewarding archive  $\mathcal{A}_{Rew}$ , trained AE;

$\mathcal{A}_{Nov} = \emptyset, \mathcal{A}_{Rew} = \emptyset, \mathcal{Q}_{Em} = \emptyset, \mathcal{Q}_{Cand.Nov} = \emptyset, \mathcal{Q}_{Cand.Em} = \emptyset, D = 0, TI_C = 0$ ;

Sample initial population  $\Gamma_0$ ;

Split  $Bud$  in chunks of size  $K_{Bud}$ ;

**while**  $Bud$  not depleted **do**

**if**  $\Gamma_0$  **then**

$Eval(\theta_i), \forall \theta_i \in \Gamma_0$ ;  $\backslash\backslash$ Evaluate initial population

$b_i = \psi(\theta_i) \in \mathcal{B}, \forall \theta_i \in \Gamma_0$ ;  $\backslash\backslash$ Calculate behavior descriptor

$Exploration(K_{Bud}, m, \sigma, \mathcal{A}_{Nov}, \mathcal{Q}_{Cand.Em}, \Gamma_g, Q, AE)$ ;  $\backslash\backslash$ Alg. 2

$TI_C = TI_C + 1$ ;  $\backslash\backslash$ Increase training interval counter

/\* Update autoencoder and descriptors \*/

**if**  $TI_C == TI$  **then**

$DS = Extract\_Dataset(\mathcal{A}_{Nov}, \mathcal{A}_{Rew}, \Gamma_g, \Gamma_g^m)$ ;

$Train\_Autoencoder(AE, DS)$ ;

$Update\_Descriptors(AE, \Gamma_g, \Gamma_g^m, \mathcal{A}_{Nov}, \mathcal{A}_{Rew}, \mathcal{Q}_{Em}, \mathcal{Q}_{Cand.Nov}, \mathcal{Q}_{Cand.Em})$ ;

$TI = TI + 1$ ;

$TI_C = 0$ ;

/\* If rewarding policies have been found \*/

**if**  $\mathcal{Q}_{Cand.Em}! = \emptyset$  **or**  $\mathcal{Q}_{Em}! = \emptyset$  **then**

$Exploitation(K_{Bud}, \mathcal{Q}_{Cand.Em}, \lambda, m, \mathcal{Q}_{Em}, \mathcal{A}_{Nov}, \mathcal{A}_{Rew}, M_{\mathcal{E}})$ ;  $\backslash\backslash$ Alg. 3

---

The search is then driven by using the information extracted by the AE from the observations collected during the evaluations of the policies. The encoder part of the AE can in fact be used as *observation function* and its *latent feature space*  $\mathcal{F}$  as  $\mathcal{B}$ . The behavior descriptor of a policy is obtained by sampling multiple high-dimensional observations along its trajectory and use the AE to extract a compressed representation. Moreover, as for the first iterations of the search the AE representation does not properly represent the behavior space yet (Grillotti and Cully, 2022), the training happens more frequently. Once a few training iterations have been performed, the AE can better represent the behaviors, so the training happens less and less frequently. A detailed description of the training process of the AE is given in Sec. 3.2.

The second important aspect of STAX is the optimization of the reward through emitters. If, during the exploration phase, a policy  $\theta_i$  obtains a reward, it will be used during the exploitation phase to instantiate an emitter in order to improve on the reward. The rationale is that behaviors similar to the rewarding behavior  $\psi(\theta_i)$  are likely rewarding too, with possibly even higher performances than  $\psi(\theta_i)$ . These behaviors can be considered belonging to the subspace of rewarding behaviors  $\mathcal{B}_{Rew} \in \mathcal{B}$  and their corresponding policies can be discovered by performing local search around  $\theta_i$  through emitters. Note that the reward exploitation performed during this phase does not rely on any behavior descriptor. The quality of  $\mathcal{B}$  learned representation then does not interfere with the reward optimization process. This means that if a reward is discovered at the initial stages of the search, when behavior space has not been learned yet, STAX can still exploit it thanks to descriptor-less emitters, without loss of performance. The details of the reward optimization are given in Sec. 3.3.

During its operation, STAX tracks the policies generated in the different phases of the search through the following buffers and containers:



- *novelty archive*  $\mathcal{A}_{\text{Nov}}$ : a collection of policies with diverse behaviors found during the *exploration phase*. One of the two collections of policies returned as outputs of STAX;
- *reward archive*  $\mathcal{A}_{\text{Rew}}$ : a collection of the most rewarding policies found during the *exploitation phase*. Other collection of policies returned as outputs of STAX;
- *candidates emitter buffer*  $\mathcal{Q}_{\text{Cand.Em}}$ : a buffer in which the rewarding policies  $\psi(\theta_i) \in \mathcal{B}_{\text{Rew}}$  found during the *exploration phase* are stored before being used to initialize emitters in the *exploitation phase*;
- *emitter buffer*  $\mathcal{Q}_{\text{Em}}$ : a buffer in which the initialized emitters to be evaluated during the *exploitation phase* are stored;
- *novelty candidates buffer*  $\mathcal{Q}_{\text{Cand.Nov}}$ : an emitter-specific buffer in which the most novel policies found by the emitter are stored. Each emitter has its own novelty candidate buffer from which policies are sampled to be added to  $\mathcal{A}_{\text{Nov}}$  at the termination of the emitter itself.

A high-level overview of the interactions between the buffers and containers is shown in Fig. 2.

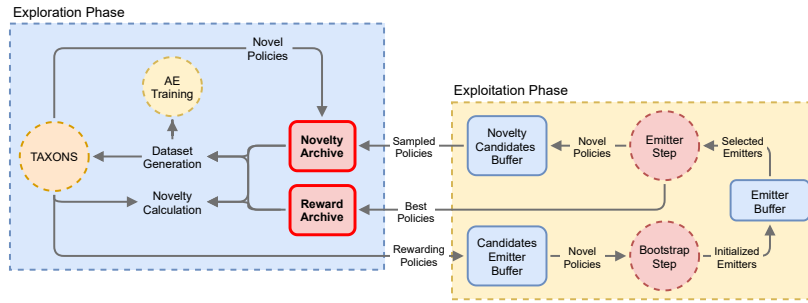


Figure 2: Overview of the containers used during the search by STAX to track the discovered policies and the initialized emitters. The two collections returned as outputs of the algorithms are highlighted in red.

The three main steps of STAX - exploration, training of the AE and exploitation of the reward - are detailed respectively in sections 3.1, 3.2 and 3.3. The whole STAX algorithm is illustrated in Fig. 1 and described in Alg. 1.

### 3.1 Exploration

Having minimal prior information about the task, STAX starts by performing the exploration step. The first time this step is performed, the parameters  $\theta \in \Theta$  of the  $M$  policies in the initial population  $\Gamma_0$  randomly sampled from the normal distribution  $\mathcal{N}(0, \mathbb{I})$ , as the policies are Neural Networks (NN). Note that any parametric function  $f(s|\theta) = a$ , where  $s$  is the state of the system and  $a$  is the action, can be used as policy. The weights of the AE used to drive the search are also randomly sampled. At each generation  $g$ ,  $m$  policies  $\theta_i^g$  are generated for each policy  $\theta_i$  in the current population  $\Gamma_g$  through a *mutation operator*. This will result in an offspring population  $\Gamma_g^m$  of size  $m \times M$  whose policies are formed as:

$$\forall j \in \{1, \dots, m\}, \forall i \in \{1, \dots, M\}, \theta_i^j = \theta_i + \epsilon, \text{ with } \epsilon \sim \mathcal{N}(0, \sigma I). \quad (3)$$

The policies in the offspring population  $\theta \in \Gamma_g^m$  are then evaluated. During the evaluation of a policy  $\theta_i$  the system traverses a trajectory of states  $\tau_s^i = [s_0^i, \dots, s_T^i]$  that are observed through

---

**Algorithm 2:** STAX Exploration Phase
 

---

**INPUT:** budget chunk  $K_{Bud}$ , number of offspring per parent  $m$ , mutation parameter  $\sigma$ , novelty archive  $\mathcal{A}_{Nov}$ , candidate emitters buffer  $\mathcal{Q}_{Cand.Em}$ , population  $\Gamma_g$ , number of policies  $N_Q$ , autoencoder AE;

**while**  $K_{Bud}$  not depleted **do**

- Generate offspring  $\Gamma_g^m$  from population  $\Gamma_g$ ;
- /\* Loop over the policies in the population \*/
- for**  $\theta_i \in \Gamma_g^m$  **do**
- $Eval(\theta_i)$ ; \\Evaluate policy
- $b_i = \psi(\theta_i) = [\dots, E(o_{t_k}^i), \dots, E(o_{t_K}^i)]$ ; \\Calc. behavior descr.
- for**  $\theta_i \in \Gamma_g^m$  **do**
- $\eta(\theta_i) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(b_i, b_j)$ ; \\Calculate novelty
- $s(\theta_i) = \sum_{k \in K} \|o_{t_k}^i - D(E(o_{t_k}^i))\|^2$ ; \\Calculate surprise
- /\* If the policy has a rewarding behavior \*/
- if**  $\psi(\theta_i) \in \mathcal{B}_{Rew}$  **then**
- $\mathcal{Q}_{Cand.Em} \leftarrow \theta_i$ ; \\Store rewarding policy
- $\mathcal{A}_{Nov} \leftarrow \text{Sample}(\Gamma_g^m, N_Q)$ ; \\Store most novel  $N_Q$  policies
- /\* NSGA-II policy selection wrt novelty and surprise \*/
- Calculate non dominated fronts  $F_j, \forall \theta_i \in \Gamma_g^m \cup \Gamma_g$ ;
- Sort fronts according to *non domination*;
- Generate  $\Gamma_{g+1}$  from most non dominated solutions  $\theta_i \in F_j$ ;
- if** *last front  $F_J$  is partially selected* **then**
- Calculate *crowding distance*  $\forall \theta_i \in F_j$ ;
- Complete filling up  $\Gamma_{g+1}$  with less crowded solution  $\theta_i \in F_j$ ;

---

sensors, generating a corresponding trajectory of observations  $\tau_o^i = [o_0^i, \dots, o_T^i]$ . The policy is then assigned a behavior descriptor  $b_i$  obtained by using *multiple observations* sampled along  $\tau_o^i$ . The descriptor is generated by encoding the sampled observations thanks to the AE's encoder  $E(\cdot)$  and then stacking their low-dimensional representations together. This can be described as  $b_i = \psi(\theta_i) = [\dots, E(o_{t_k}^i), \dots, E(o_{t_K}^i)]$ , where  $o_{t_k}^i$  is the observation generated by the policy  $\theta_i$  at time-step  $t_k$ . Sampling multiple observations along the trajectory is in contrast to what Paolo et al. (2020) did in TAXONS, in which only the last observation was used to generate the descriptor. Using the last observation in fact requires such observation to be informative of the behavior of the policy over the whole trajectory. On the contrary, using multiple observations along  $\tau_s$ , such an assumption is not required anymore.

The diversity of a policy is evaluated through two metrics: *novelty* and *surprise*. The first one is the normalized euclidean distance in the learned  $\mathcal{B}$ :

$$\eta(\theta_i) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(\psi(\theta_i), \psi(\theta_j)). \quad (4)$$

At the same time, the surprise is calculated as the *sum* of the AE's *reconstruction error* over each one of the sampled observations generated by  $\theta_i$ . A higher surprise implies that the AE has not seen that area of the learned behavior space very often. This means that selecting policies with high surprise leads the algorithm to increased exploration. Such metric is defined as:

$$s(\theta_i) = \sum_{k \in K} \|o_{t_k}^i - D(E(o_{t_k}^i))\|^2, \quad (5)$$

where  $K$  is the list of indexes of the selected time-steps along the trajectory.

The two metrics are used to select the policies that will form the population for the next generation  $\Gamma_{g+1}$  through the NSGA-II multi-objective approach (Deb et al., 2002). This is in contrast to what was done by Paolo et al. (2020) in TAXONS, in which only one among novelty and surprise was sampled at each generation to be used for policy selection.

Finally, STAX samples uniformly  $N_Q$  policies to be added to the *novelty archive*  $\mathcal{A}_{\text{Nov}}$ . Moreover, all the rewarding policies found in this phase are added in the *candidates emitter buffer*  $\mathcal{Q}_{\text{Cand.Em}}$  to be used during the *exploitation phase* to generate emitters. The whole exploration process is shown in Algorithm 2.

### 3.2 Training of the autoencoder

The exploration performed by STAX is driven by the AE. This means that the way the AE itself is trained, and thus the quality of the learned low-dimensional representation, is fundamental in order to obtain good exploration. In order to meaningfully look for diversity in the learned behavior space  $\mathcal{B}$ , the AE has to be trained on the data collected during the search for policies itself. This data is collected into a dataset  $DS$  consisting of the observations used to generate the behavior descriptor of the policies, as defined in Sec. 3.1. The policies whose observations are added to  $DS$  are the ones contained in both the *reward archive*  $\mathcal{A}_{\text{Rew}}$  and the *novelty archive*  $\mathcal{A}_{\text{Nov}}$ , with the addition of the observations from the population  $\Gamma_g$  and the offspring population  $\Gamma_g^m$  of the last evaluated generation  $g$ . The data of the archives provides a *curriculum*, stabilizing the training process and preventing the search from cycling back to already explored areas. At the same time, adding the observations from the most recent population to the training dataset helps the AE to better represent the frontier of the explored space, towards which the search is to be pushed.

Once the dataset  $DS$  has been collected, it is split into two sub-datasets: the *training dataset*  $DS_{\text{Train}}$  and the *validation dataset*  $DS_{\text{Val}}$ . For each training episode, the AE is trained on the  $DS_{\text{Train}}$ . At the end of each training epoch on  $DS_{\text{Train}}$ , the model validation error is calculated on  $DS_{\text{Val}}$ . The training episode is stopped if the error increases for 3 consecutive epochs.

As stated in Sec. 3, the AE is trained less frequently the longer the search is performed; the same strategy is employed in the AURORA method (Cully, 2019). This allows adapting the frequency of the training to the maturity of the learned  $\mathcal{B}$ , while saving time and computational resources with respect to training the AE at fixed intervals. This shifting training regime is obtained by performing the training process every  $TI$  exploration steps. At the beginning of the search, STAX sets  $TI = 1$ . Its value is then increased by 1 every time a training episode is performed. Finally, at the end of each training episode, the behavior descriptor of all the policies present in the archives and in the populations is updated with the new descriptors generated by the retrained AE. This keeps the behavior descriptors and the novelty measurements of the policies consistent and meaningful.

### 3.3 Exploitation

At the end of the exploration step, if the *emitters candidate buffer*  $\mathcal{Q}_{\text{Cand.Em}}$  or the *emitters buffer*  $\mathcal{Q}_{\text{Em}}$  are not empty, the meta-scheduler assigns a budget chunk  $K_{\text{Bud}}$  to the exploitation step. The objective of this phase is to optimize the reward. In practice, this consists of i) identifying the policies that can be used to initialize the populations of the emitters and ii) running such emitters with the goal of generating solutions with high rewards. This is done through two sub-steps: the *bootstrap step* and the *emitter step*. During the bootstrap step, the rewarding policies collected in the *emitters candidate buffer*  $\mathcal{Q}_{\text{Cand.Em}}$  are used to instantiate emitters that are then evaluated for few iterations. The emitter with the potential to improve on the reward are added to the *emitter buffer*  $\mathcal{Q}_{\text{Em}}$  to be fully evaluated during the subsequent emitter step. At the same time, the emitters not capable of improving on the reward are discarded. In the

following sub-step, the emitters from  $\mathcal{Q}_{\text{Em}}$  are sampled according to their performance and evaluated until termination or until the budget chunk is depleted.

Such a two sub-steps process allows STAX to quickly decide which of the rewarding policies from  $\mathcal{Q}_{\text{Cand.Em}}$  is worth optimizing and which is not. This prevents the waste of computational budget on the optimization of policies that are in hard-to-escape local optima of the reward landscape. Moreover, using emitters allows to disjointly optimize multiple reward areas in an efficient way by quickly finding good solutions. This is fundamental for an approach like STAX in which the  $\mathcal{B}$  is autonomously learned. In hand-designed  $\mathcal{B}$  the engineer has total control over the space itself, allowing him to reduce the disjointedness of the reward areas. This is not the case when the behavior descriptor is generated by stacking multiple learned representations extracted from high-dimensional observations, as done by STAX. In this kind of setting, there is no guarantee that the new space will have the same structure of the reward areas as the ground-truth hand-designed  $\mathcal{B}$ . Given the complex nature of the learned space, due to the stacking of the encoding of multiple observations, it can happen that this space contains multiple reward areas, even if only one is present in the ground-truth  $\mathcal{B}$ . For these reasons, using an emitter-based approach as STAX capable of focusing on multiple reward areas can give a strong advantage in situations where the behavior representation is so complex.

In the following, we will describe in detail first how our emitters work and how the two sub-steps of the exploitation process use them to optimize the reward. The whole exploitation phase is detailed in Algorithm 3.

### Emitters

Emitters are what STAX uses to optimize the reward. An emitter in an instance of a *reward-based EA*. While any reward optimization method can be used, in this paper we base the emitters on an *elitist EA*, similarly to the work of Paolo et al. (2021). At each generation, the emitter selects the population among the best performing policies  $\tilde{\theta}_j$  from the previous generation's population and offsprings, while the offsprings themselves are generated according to Eq. (3). Using an elitist EA removes the need to estimate a covariance matrix from the emitter population. This estimation can be unstable in situations in which the population size is lower than the dimensionality of the space, as can be often the case when working with neural networks. To prevent this instability issue, methods like CMA-ES (Hansen, 2016) take into account information about older generations when estimating the covariance. This can render the estimation of the quality of an emitter from its initial generations less reliable, limiting the performance of a method like STAX which discards less promising emitters according to their initial performance.

Each one of the emitters  $\mathcal{E}_i$  used by STAX consist of a population  $P_\gamma$  of size  $M_\mathcal{E}$  of policies  $\tilde{\theta}_i \in \Theta$ , its offspring population  $P_\gamma^m$  of size  $m \times M_\mathcal{E}$ , a *novelty candidates buffer*  $\mathcal{Q}_{\text{Cand.Nov}}$  in which the most novel policies are stored, a generation counter  $\gamma$ , and a tracker for the highest reward found until now  $R_\gamma$ . At the same time, the emitter also tracks two novelties,  $\eta_\gamma$ , that is the novelty of the most novel policy found until generation  $\gamma$ , and the *emitter novelty*,  $\eta(\mathcal{E}_i)$ , corresponding to the novelty of the policy generating the emitter. The emitter is initialized from the policy  $\theta_i$  by sampling the  $M_\mathcal{E}$  policies in its initial population  $P_0$  from the distribution  $\mathcal{N}(\theta_i, \sigma_i I)$ . To reduce the overlap of the emitter's search space with the ones of possible nearby emitters, STAX shapes  $\mathcal{N}(\theta_i, \sigma_i I)$  such that the distance between  $\theta_i$  and the closest  $\theta_j$  corresponds to 3 standard deviations. This is done by initializing  $\sigma_i$  as:

$$\sigma_i = \frac{\min_j(\text{dist}(\theta_i, \theta_j))}{3}, \quad \forall \tilde{\theta}_j \in \Gamma_g^m \cup \tilde{\Gamma}_g. \quad (6)$$

During its evaluation, an emitter tracks also its own *emitter improvement*  $I(\mathcal{E}_i)$ , a metric that is then used by STAX to select which emitters to prioritize and which to discard, allowing a better

**Algorithm 3:** STAX Exploitation Phase

---

**INPUT:** budget chunk  $K_{Bud}$ , candidate emitters buffer  $\mathcal{Q}_{Cand\_Em}$ , number of bootstrap generations  $\lambda$ , emitter population size  $M_{\mathcal{E}}$ , number of offspring per policy  $m$ , emitters buffer  $\mathcal{Q}_{Em}$ , rewarding archive  $\mathcal{A}_{Rew}$ , novelty archive  $\mathcal{A}_{Nov}$ ;

```

/* Bootstrap step */
while  $K_{Bud}/3$  not depleted do
  Select most novel policy  $\theta_i$  from  $\mathcal{Q}_{Cand\_Em}$ ;
  \Calculate emitter's mutation standard deviation
   $\sigma_i = \min_j(\text{dist}(\theta_i, \theta_j))/3, \forall \theta_j \in \Gamma_g^m \cup \tilde{\Gamma}_g$ ;
  Initialize:  $\mathcal{E}_i, \mathcal{Q}_{Cand\_Nov}^i = \emptyset$ , and  $P_0$ ;
  for  $\gamma \in \{0, \dots, \lambda\}$  do
    if  $P_0$  then
      |  $Eval(\tilde{\theta}_j), \forall \tilde{\theta}_j \in P_0$ ; \Evaluate initial population
      Generate offspring population  $P_{\gamma}^m$  from  $P_{\gamma}$ ;
       $Eval(\tilde{\theta}_j), \forall \tilde{\theta}_j \in P_{\gamma}^m$ ; \Evaluate offspring population
      Generate  $P_{\gamma+1}$  from best  $\tilde{\theta}_j \in P_{\gamma}^m \cup P_{\gamma}$ ;
    Calculate improvement  $I(\mathcal{E}_i)$ ;
    /* Store promising emitters in emitters buffer */
    if  $I(\mathcal{E}_i) > 0$  then
      |  $\mathcal{Q}_{Em} \leftarrow \mathcal{E}_i$ ;
  /* Emitters step */
  Calculate pareto fronts in  $\mathcal{Q}_{Em}$ ;
  while  $2/3 K_{Bud}$  not depleted do
    Sample  $\mathcal{E}_i$  from non-dominated emitters in  $\mathcal{Q}_{Em}$ ;
    while not terminate( $\mathcal{E}_i$ ) do
      Generate offspring population  $P_{\gamma}^m$  from  $P_{\gamma}$ ;
       $Eval(\tilde{\theta}_j), \forall \tilde{\theta}_j \in P_{\gamma}^m$ ; \Evaluate population
       $\mathcal{A}_{Rew} \leftarrow \tilde{\theta}_j, \forall \tilde{\theta}_j \in P_{\gamma}^m \mid r(\tilde{\theta}_j) > R_{\gamma}$ ; \Store high rewarding
      policies
       $\mathcal{Q}_{Cand\_Nov}^i \leftarrow \tilde{\theta}_j, \forall \tilde{\theta}_j \in P_{\gamma}^m \mid \eta(\tilde{\theta}_j) > \eta_i$ ; \Store high novelty
      policies
      Generate  $P_{\gamma+1}$  from best  $\tilde{\theta}_j \in P_{\gamma}^m \cup P_{\gamma}$ ;
      Update  $I(\mathcal{E}_i)$  and  $R_{\gamma}$ ;
      if terminate( $\mathcal{E}_i$ ) then
        |  $\mathcal{A}_{Nov} \leftarrow \text{Sample}(\mathcal{Q}_{Cand\_Nov}^i, N_Q)$ ; \Store  $N_Q$  novel policies
        | in archive
        Discard emitter  $\mathcal{E}_i$ ;

```

---

allocation of evaluation budget. A positive  $I(\mathcal{E}_i)$  means that the emitter can improve on its initial reward. On the contrary,  $I(\mathcal{E}_i) \leq 0$  means that the chances for the emitter to find better rewards are low, so it is not worth allocating more evaluation budget to it.

The improvement is calculated as the difference between the average reward obtained during the most recent and the initial generations of the emitter:

$$I(\mathcal{E}_i) = \frac{1}{\lambda M_{\mathcal{E}}} \left( \sum_{\gamma=T-\lambda/2}^T \sum_{j=0}^{M_{\mathcal{E}}} r(\gamma, j) - \sum_{\gamma=\gamma_0}^{\lambda/2} \sum_{j=0}^{M_{\mathcal{E}}} r(\gamma, j) \right), \quad (7)$$

where  $T$  is the last evaluated generation,  $r(\gamma, j)$  is the reward of policy  $\tilde{\theta}_j \in P_{\gamma}$  and  $\gamma_0$  is the

generation at which the emitter is at the beginning of its evaluation.

An emitter is terminated once a *termination criterion* is reached. There can be many termination criteria, depending on the kind of algorithm used as emitter. In this work, we use the one introduced by Paolo et al. (2021). This criterion is directly inspired by the *stagnation criterion* used for the CMA-ES algorithm and introduced by Hansen (2016) and stops the emitter once there is no more improvement on the reward. This is calculated by tracking the history of the rewards obtained by the emitter over the last  $120 + 20 * n / M_{\mathcal{E}}$ , where  $n$  is the size of the parameter space  $\Theta$  and  $M_{\mathcal{E}}$  is the population size of the emitter. The termination condition is met if the *maximum* or the *median* of the last 20 rewards is lower than the *maximum* or the *median* of the first 20 rewards.

### Bootstrap step

The candidate emitters buffer  $\mathcal{Q}_{\text{Cand.Em}}$  contains all the rewarding policies found during the exploration phase. During the bootstrap step, emitters are initialized from these policies starting from the most novel ones with respect to the reward archive  $\mathcal{A}_{\text{Rew}}$ . This allows STAX to focus more on the less explored areas of the rewarding behavior space  $\mathcal{B}_{\text{Rew}}$ .

Once an emitter  $\mathcal{E}_i$  has been initialized, it is executed for  $\lambda$  generations to evaluate its potential for improving the reward by calculating its initial *emitter improvement*  $I(\mathcal{E}_i)$ . Only the emitters with positive improvement after this initial evaluation phase are added to the *emitter buffer*  $\mathcal{Q}_{\text{Em}}$  for further evaluation during the emitter step, while the rest are discarded. This allows STAX to quickly discard emitters whose initializing policy is in a hard-to-optimize local minima of the reward space. At the same time, it helps in discovering the policies whose behaviors are in the most promising regions of the rewarding behavior space  $\mathcal{B}_{\text{Rew}}$ . The whole bootstrap step lasts  $K_{\text{bud}}/3$  evaluation steps, at the end of which STAX switches to the *emitter step*.

### Emitter step

During this step, STAX evaluates the emitters that, due to a positive *emitter improvement*, are now present in the *emitter buffer*  $\mathcal{Q}_{\text{Em}}$ . The step starts by calculating the Pareto front between the improvement  $I(\cdot)$  and the emitter novelty  $\eta(\cdot)$  of the emitters in the buffer. The emitter  $\mathcal{E}_i$  to run is then sampled from the front of the *non-dominated emitters*. This allows STAX to focus on the most promising and less explored areas of the rewarding search space  $\mathcal{B}_{\text{Rew}}$ .

The policies  $\tilde{\theta}_i$  found during the evaluation of an emitter  $\mathcal{E}_i$  can be stored either for their novelty or for the reward they obtain. At every generation  $\gamma$ , the policies with a novelty higher than the maximum novelty found by the emitter so far,  $\eta_{\gamma-1}$ , are stored in the *novelty candidates buffer*  $\mathcal{Q}_{\text{Cand.Nov}}$ . At the same time, the policies with a reward higher than the maximum reward found until  $\gamma - 1$ ,  $R_{\gamma-1}$ , are stored into the *reward archive*  $\mathcal{A}_{\text{Rew}}$ . Once these policies have been stored, both  $\eta_{\gamma-1}$  and  $R_{\gamma-1}$  are updated with the new maximum values.

The emitter  $\mathcal{E}_i$  is run until either one of these two conditions happens: the  $2/3 K_{\text{Bud}}$  evaluation budget chunk is depleted or a termination condition is met. The first case leads STAX to update the improvement of  $\mathcal{E}_i$  and store it again in the emitters buffer  $\mathcal{Q}_{\text{Em}}$  for a possible future evaluation. The algorithm then goes back to the exploration phase. In the second case, the emitter is terminated and  $N_Q$  policies from the emitter's novelty candidate buffer are uniformly sampled to be added to the novelty archive  $\mathcal{A}_{\text{Nov}}$ . This allows STAX to save particularly novel solutions to  $\mathcal{A}_{\text{Nov}}$  and prevent the search to go back to already explored areas. Finally, a new emitter to be evaluated is selected from the front of non-dominated emitters.

## 4 Experiments

This section studies how STAX can discover highly rewarding policies while exploring a behaviour representation space learned online. All of this with minimal previous information about the environment and the task at hand. STAX will be compared against various baselines.

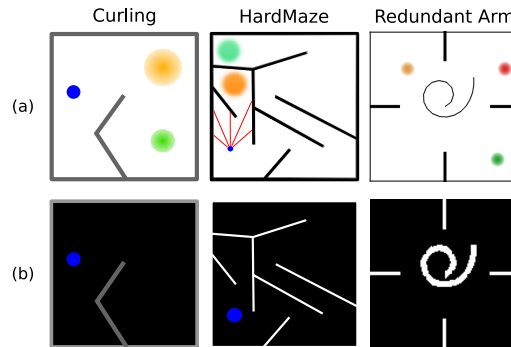


Figure 3: The three testing environments. Row (a) shows the real environments. The reward areas are represented by the green, orange and red circles. Row (b) contains the  $64 \times 64$  RGB observations of the environment as seen by the AE. The behavior descriptors are generated by sampling 5 of these images along the trajectories.

Moreover, multiple ablation studies will be performed to study which aspects of the method are the most important ones. In Sec. 5.1 we will evaluate the exploration performance of the algorithms, while the exploitation performance will be studied in Sec. 5.2. An example of the final distribution of the behavior representation learned by the discovered policies is given in Sec. 5.3. Ablation studies on the factors contributing to the exploration performance and the importance of the AE training regime are done respectively in Sec. 5.4 and Sec. 3.2. Finally, in Sec. 5.6 we evaluate the quality of the learned  $\mathcal{B}$ .

In order to perform this analysis, STAX is evaluated on 3 sparse rewards environments, shown in Fig. 3.

**Curling:** it consists of a 2 Degrees of Freedom (DoF) arm pushing a ball over a table (Paolo et al., 2021). The arm is controlled by a 3 layers NN with each layer of size 5. The input of the controller is a 6-dimensional array containing the  $(x, y)$  ball pose and the joints angles and velocities. The controller outputs a 2-dimensional array containing the speeds of the two joints at the next time-step. Each policy is run in the environment for 500 timesteps. The reward is given only if the ball is in one of the two rewarding areas and is higher the closer it is to the center of the area. The ground truth behavior descriptor used by methods that do not learn the representation is the final  $(x, y)$  position of the ball. The environment, together with the  $64 \times 64$  RGB image the AE sees during the algorithm execution, is shown in Fig. 3.

**HardMaze:** it consists of a 2-wheeled robot whose goal is to navigate a maze with the aid of 5 distance sensors (Lehman and Stanley, 2008). The robot, in blue in Fig. 3, is controlled by a 2-layers NN with each layer of size 5. The controller receives as inputs the reading of the 5 distance sensors, shown in red in Fig. 3, and outputs the speed of the wheels for the next timestep. The agent receives a reward if the robot reaches one of the 2 reward areas, with the reward being higher the closer the robot stops. Each policy is run in the environment for 2000 timesteps. The ground truth behavior descriptor used by methods that do not learn the representation is the final  $(x, y)$  position of the robot. The environment, together with the  $64 \times 64$  RGB image the AE sees during the algorithm execution, is shown in Fig. 3.

**Redundant Arm:** it consists of a 20-DoF arm moving on a 2-dimensional plane (Loviken and Hemion, 2017). The arm is controlled by a NN with 2 layers, each one of size 5. The input of the controller is the 20-dimensional vector of each joint's positions, while the output consists of the 20-dimensional joints' torque vector. The policies are run for 100 timesteps each, or until the arm collides either with the wall or itself. The ground truth behavior descriptor used by methods that do not learn the representation is the final  $(x, y)$  position of the end effector. The

reward is given if the end effector reaches one of the three highlighted areas, with the reward being higher the closer the effector is to the center of the reward area. The environment, together with the  $64 \times 64$  RGB image the AE sees during the algorithm execution, is shown in Fig. 3.

In all these environments, STAX builds the behavior descriptors by stacking the low-dimensional representations extracted by the AE from multiple high-dimensional observations. To this end, 5 samples collected at regular intervals along the trajectories are used during the experiments.

### Baselines

STAX is compared against the following baselines:

- **NS** (Lehman and Stanley, 2008): vanilla NS, that performs pure exploration in the ground-truth behavior space and does not attempt to improve on the reward;
- **MAP-Elites (ME)** (Mouret and Clune, 2015): vanilla MAP-Elites that uses a  $50 \times 50$  grid to cover the ground-truth behavior space of each environment;
- **MOO-NR** (Deb et al., 2002): a multi-objective evolutionary algorithm optimizing both the novelty in the ground-truth behavior space and the reward of the policies;
- **TAXONS** (Paolo et al., 2020): that performs pure exploration by learning the behavior descriptor through an AE trained during the search process;
- **SERENE** (Paolo et al., 2021): that performs exploration through NS in the ground-truth behavior space, exploiting any discovered reward through emitters.

Note that among the baselines, only TAXONS learns the behavior descriptor similarly to STAX. The other baselines all use the ground-truth behavior descriptor.

For each experiment, the given evaluation budget is  $Bud = 500000$ , with a chunk size of  $K_{Bud} = 100$ . The population has a size of  $M = 100$  and each policy generates  $m = 2$  offsprings. This is done by using a mutation parameter of  $\sigma = 0.5$ . At each generation, the number of policies sampled to be added to the novelty archive is  $N_Q = 5$ . The emitters have a population size of  $M_E = 6$  with a bootstrap phase of  $\lambda = 6$ . For every experiment, the policies' parameters are bounded in the  $[-5, 5]$  range. All approaches using an AE to represent the behavior descriptor use the same structure. The AE consists of an encoder  $E(\cdot)$  with 4 convolutional layers of sizes  $[32, 64, 32, 16]$ , followed by a linear layer projecting the 256-dimensional vector returned by the last convolutional layer into the 10-dimensional feature space. Each convolutional operation has a kernel of size 4, with a stride of 2 and a padding of 1. Every layer is followed by a SeLU activation function (Klambauer et al., 2017), allowing the self-normalization of the NN. On the contrary, the decoder  $D(\cdot)$  starts with a linear layer projecting the 10-dimensional feature vector into a 256-dimensional vector. Then it is followed by 4 convolutional layers of sizes  $[32, 64, 32, 3]$ , each one using a kernel of size 4, a stride of 2, and a padding of 1. Every layer uses a SeLU activation function, with the exception of the last convolutional one using a ReLU, in order to force the non-negativity of the output value. The weights of the AE are randomly initialized through the default Pytorch initialization. This is done by sampling the weights of each layer from a uniform distribution  $U(-\frac{1}{\sqrt{\omega}}, \frac{1}{\sqrt{\omega}})$ , with  $\omega$  being the number of learned parameters in the layer. The training is done with the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 0.001. The results are computed over 15 runs for each experiment and their statistical significance is evaluated by performing a Mann-Whitney test (Mann and Whitney, 1947) with Helm-Bonferroni correction (Holm, 1979). Finally, in each plot, the performances of methods using the ground-truth  $\mathcal{B}$  are represented with dashed lines, while the



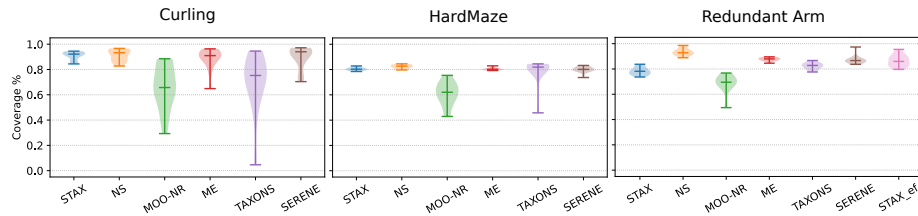


Figure 4: Final coverage reached by STAX against the different baselines after  $5 \times 10^5$  evaluations. The medians and the extrema are highlighted. The plots are calculated over 15 seeds.

methods learning the behavior space are shown through a continuous line. The code repository is available at: <https://github.com/GPaolo/STAX.git>.

## 5 Results

In this section, the results obtained from the experiments are discussed. The significance of the results is evaluated through the non-parametric Mann-Whitney U test (Mann and Whitney, 1947) with Holm-Bonferroni correction (Holm, 1979).

### 5.1 Exploration

This section studies how well STAX can explore in situations of sparse rewards while having minimal information about the environment and the task. This is done by measuring the *coverage metric* obtained in the *ground truth*  $\mathcal{B}$  defined in Sec. 4 for each one of the tested environments. The coverage metric is evaluated by dividing said ground truth space into a  $50 \times 50$  grid and calculating the percentage of cells occupied during the search. A cell is considered occupied if a policy reaches it at the end of its evaluation. Note that, while the coverage is calculated in the ground-truth space, STAX has no access to this space at search time. The algorithm has to learn a representation from a collection of high-dimensional observations in order to perform the exploration. This means that the method can also explore areas of the space that are not considered by the coverage metric in the ground-truth space. An example of this is the Curling environment, in which a single final position of the ball - the one considered in the ground-truth  $\mathcal{B}$  - can correspond to multiple arm positions that are represented by STAX. At the same time, the strongest baseline with respect to this metric is NS which has direct access to the space in which the coverage is calculated, providing an upper-bound value for our experiments.

Fig. 4 shows the coverage reached by our method and all the tested baselines. It can be seen that on the Curling environment STAX performs similarly to NS, with a mean final coverage of 90.8% for STAX compared to the 91% for NS ( $p = .77$ ). In the other two environments, STAX reaches lower coverage compared to NS. The difference is small on the HardMaze, 80.3% for STAX versus 82.2% for NS ( $p = 1.5 \times 10^{-2}$ ), but it is higher on the Redundant Arm environment with 78.2% for STAX against the 93.3% obtained by NS ( $p = 7.31 \times 10^{-5}$ ).

The reason for STAX's low performances on this last environment are due to STAX learning to represent the whole arm configuration rather than only the end effector position, thus maximizing diversity in dimensions not considered by the coverage metric. On the contrary, the 86.6% of coverage reached by STAX when the AE is only shown the end effector position, rather than the whole arm, STAX\_ef in the Redundant Arm plot in Fig. 4, are comparable to the coverage of 87% reached by SERENE ( $p = .47$ ). The other methods using the hand-designed ground-truth  $\mathcal{B}$  to drive the search - ME and SERENE - reach high levels of coverage comparable to NS on Curling (ME:  $p = .77$ , SERENE:  $p = .77$ ), but slightly lower on both Redundant Arm (ME:  $p = 1.7 \times 10^{-4}$ , SERENE:  $p = 5.8 \times 10^{-4}$ ) and HardMaze (ME:  $p = 1.7 \times 10^{-2}$ , SERENE:  $p = 2.6 \times 10^{-2}$ ). This is expected given that both methods perform the search in the same space in which the coverage metric is computed but also optimize the

reward. The good performance of STAX is instead obtained with minimal information about the task and the space in which information is gathered. At the same time, MOO-NR struggles in all environments, likely because once a rewarding solution is found, it will dominate all the non-rewarding solutions, strongly limiting the exploration of the method.

TAXONS also obtains high coverage, with the notable exception of the Curling environment. The culprit of this loss of performance is likely the presence of the 2-DOF arm in the image fed to the AE, as shown in Fig. 3, that can act as a distractor in situations in which only the final position of the ball is interesting. At the same time, the presence of the arm is not a hindrance to the performances of STAX. This is likely due to both the higher amount of data on which the AE is trained - the 5 frames sampled along the trajectory for STAX compared to only the last frame for TAXONS - and the better selection of new policies according to the MOO based approach, performed by STAX. The effects of these factors on the performance of STAX will be studied in Sec. 5.4.

## 5.2 Exploitation

The maximum reward achieved by the algorithms in all the reward areas is shown in Fig. 5. Using emitters to exploit the reward allows STAX to reach high rewards in a few evaluations. These performances are similar to the ones obtained by SERENE on Curling ( $p = .53$ ) and HardMaze ( $p = .71$ ) and slightly higher on Redundant Arm ( $p = 1.4 \times 10^{-2}$ ), thanks to the fact that the reward exploitation performed by the emitters does not rely on any behavior descriptor. Among the other baselines performing reward improvement, the best performing one is ME, capable of reaching high values on all reward areas, but at a much slower pace than STAX. This is not the case for the multi-objective approach MOO-NR, which can always find at least one of the multiple reward areas, but then tends to extensively focus on it, instead of also exploring other areas. For this reason, only the easiest reward area is exploited to high values in all environments, while the harder reward area is seldom exploited. On the contrary, while NS and TAXONS can perform good exploration, they cannot reach high reward levels very quickly, with TAXONS being consistently worse in this regard than any other algorithm ( $p = 2.5 \times 10^{-4}$ ). This is due to the lack of any reward-exploitation mechanism present in both methods. This is even more noticeable in the redundant arm environment, where even if TAXONS can reach higher coverage levels than STAX ( $p = 6.9 \times 10^{-3}$ ), the absence of any reward improving mechanism leads to very low performances on all reward areas.

## 5.3 Final archives distribution

An example of the final distribution of the behaviors representations for the policies in the final archives is shown in Fig. 6. Each point represents a policy. In blue are shown the policies present in the novelty archive  $\mathcal{A}_{\text{Nov}}$ , while in orange are the policies in the reward archive  $\mathcal{A}_{\text{Rew}}$ . For the baselines not using the double archives structure, the blue points represent the policies that did not receive any reward, considered *exploratory*, while the orange points represent the rewarding policies. If a method is capable of properly exploring the behavior space, the blue dots should cover as much as possible of the space. At the same time, a method capable of optimizing the reward, should be able to focus on the reward areas, thus producing many solutions reaching said areas (orange dots).

From the figure, it is possible to see how emitter-based methods, STAX and SERENE, densely cover the reward areas discovered during exploration, while NS and TAXONS do not have this effect due to the lack of any exploitation mechanism. At the same time, the row of MOO-NR shows how once reward areas are discovered, the method mainly focuses on those. Finally, the figure shows how ME very uniformly covers the search space compared to the other methods, thanks to the discretization of the behavior space.

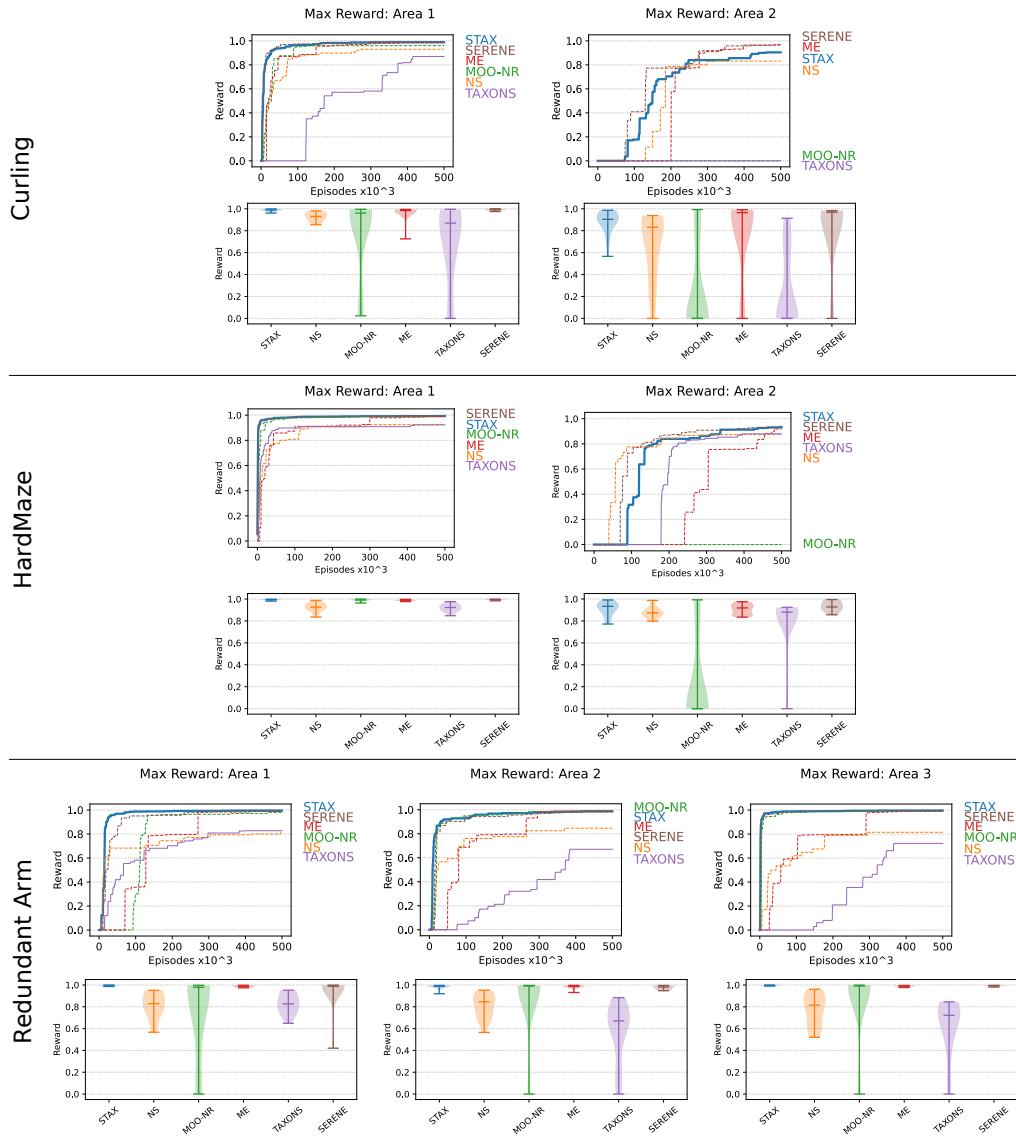


Figure 5: Maximum reward reached in all the reward areas by STAX against the different baselines. For each environment, the top row represents the median maximum reward with respect to the whole evaluation budget. The bottom row represents the final maximum reward obtained by the algorithms. The medians and the extrema are highlighted. All plots have been calculated over 15 seeds.

#### 5.4 Exploration ablation studies

This section studies the contributing factors to the exploration results obtained by STAX. The study focuses on two aspects of the algorithm: the multi-objective approach for policy selection and the multiple observations used to generate the behavior descriptor of a policy. Four ablated variants of STAX are considered:

- **STAX\_multi**: it is the vanilla version of STAX. It uses both the multi-objective policy

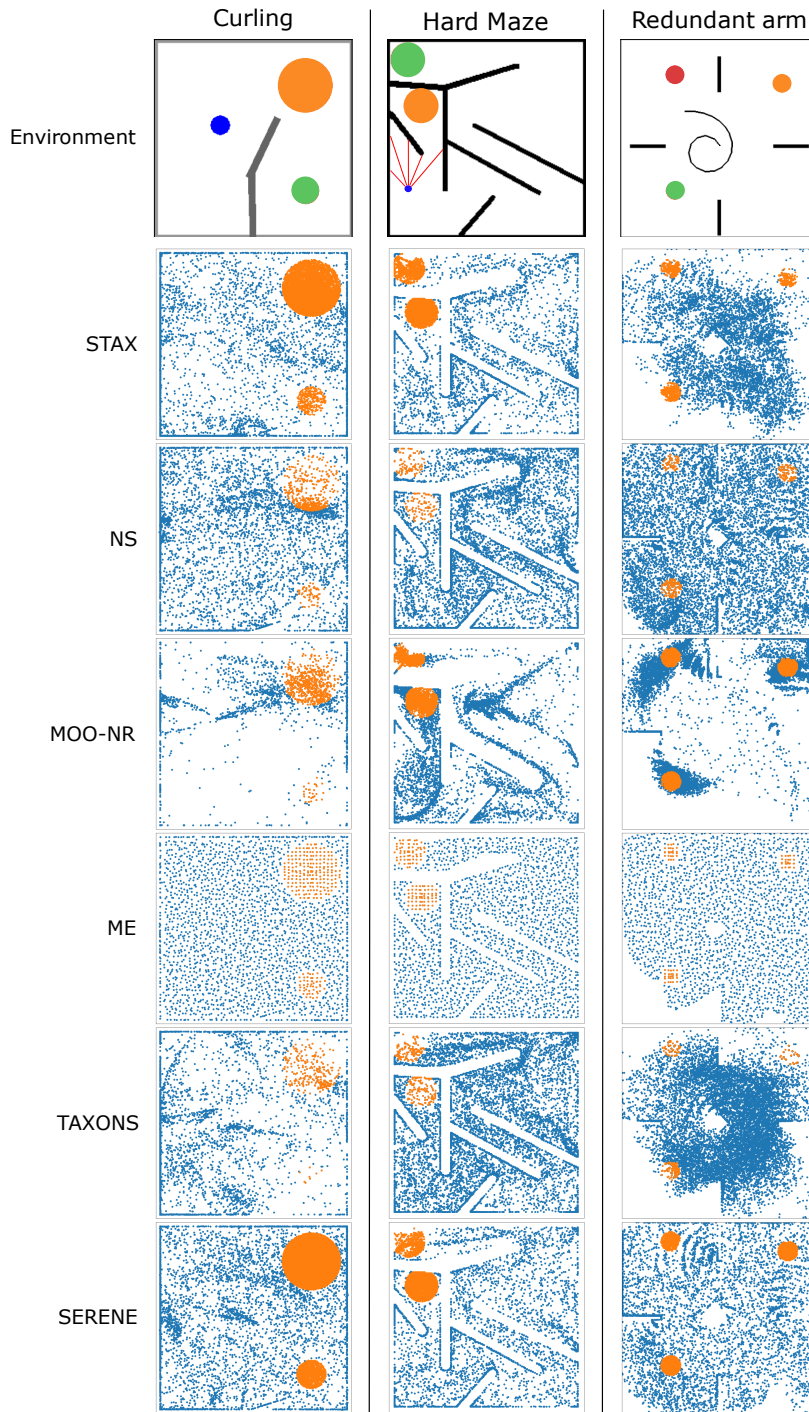


Figure 6: Distribution of the behavior descriptors of the archived policies. On each column are shown the results for an environment, while on each row is shown the distribution for each experiment. The archive plotted are from the runs achieving the highest coverage. In blue are the policies outside of the reward area, while in orange are the policies within the reward area.

selection between novelty and surprise and the 5 observations sampled along the policy trajectory to generate the behavior descriptor;

- **STAX\_single**: this variant still uses the multi-objective policy selection strategy, but the behavior descriptor is calculated only from the last observation. This baseline is used to evaluate how important is to use points along the whole trajectory rather than just the last one;
- **STAX-ALT\_multi**: this variant uses the same strategy used by TAXONS to select between novelty and surprise, sampling either one of the two at each generation. The behavior descriptor is generated by using 5 observations sampled at regular intervals along the trajectory. This baseline is used to evaluate the importance of the new policy-selection strategy used by STAX;
- **STAX-ALT\_single**: as the previous variant, here the TAXONS policy selection strategy is used. Moreover, the behavior descriptor is generated by only the last observation of the trajectory.

Both the coverage and the maximum reward reached by each variant over each reward area are analyzed.

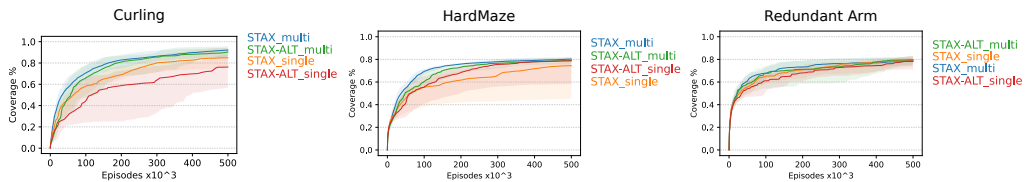


Figure 7: Median coverage with respect to the given evaluation budget reached by STAX against the ablated versions of the algorithm. The shaded areas represent the 10 and 90 percentile calculated over 15 seeds.

The average coverage is shown in Fig. 7. It is possible to see that the final reached coverage is similar for all variants on all environment ( $p > .05$ ), with the exception of the Curling environment in which the variants using multiple observations reach higher final coverage compared to both `STAX_single` ( $p < 0.028$ ) and `STAX-ALT_single` ( $p < 4.33 \times 10^{-5}$ ). Nonetheless, the variants using multiple observations reach higher coverage earlier during the runs compared to the single observation variants. This is likely due to the AEs of these variants being trained on 5 times more data than the ones of the variants using a single observation. Moreover, being the data collected along the whole trajectory, it provides a more diverse collection of data from the observation space, making it easier for the algorithm to learn a good representation.

The improved performance provided by using multiple observations can be seen also when analyzing the maximum reward reached in the environments, as shown in Fig. 8. While the final reward reached for the different reward areas is similar for all the methods, `STAX_multi` tends to reach high rewards earlier in the runs compared to the ablated approaches.

## 5.5 Autoencoder training regime

This section analyzes how the way  $\mathcal{B}$  is learned through the AE influences the search. In this regard, the study focuses on two aspects: how important it is to learn the representation versus just using a random one and if retraining from scratch the AE at each training episode has any influence on the search process. In STAX the AE is continuously trained across different training episodes. This means that similarly to what is done by Paolo et al. (2020), the training of

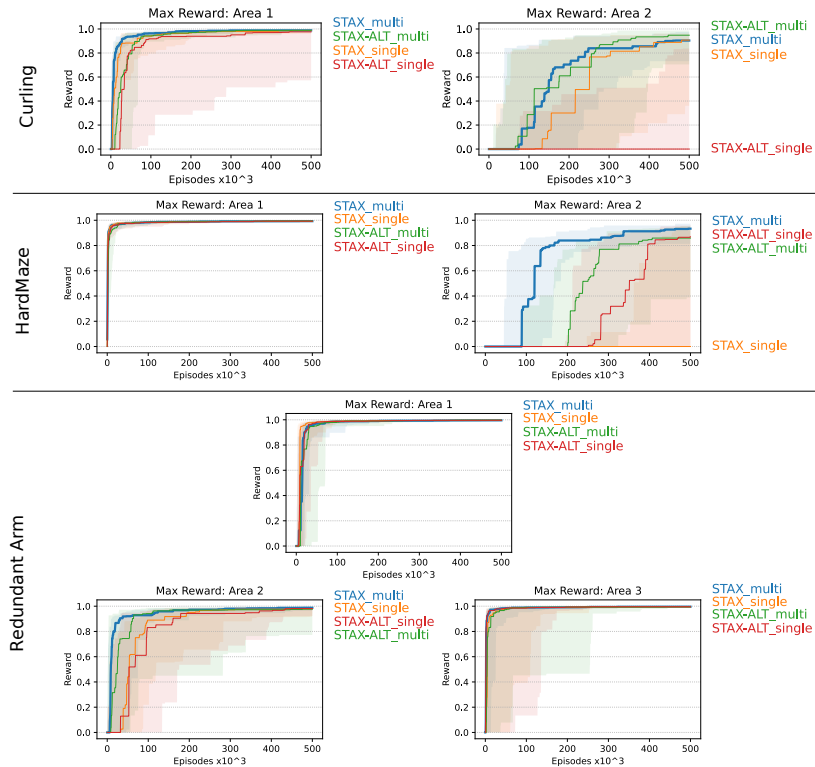


Figure 8: Median maximum reward reached in all the reward areas by STAX against the ablated versions of the algorithm. The shaded areas represent the 10 and 90 percentile calculated over 15 seeds.

the AE is resumed at every training episode. This produces a *curriculum effect* over the borders of the explored space due to the training on the last generation of the population and offsprings. The curriculum effect is also given by training the AE over the archives, even if this contribution is small at the beginning of the search when the archives contain only a few elements.

To analyze these two aspects, STAX is compared against 2 variants:

- **STAX-NT**: in which the search is driven through an AE whose weights are randomly sampled at the beginning of the search and not modified anymore;
- **STAX\_reset**: in which the weights of the AE are randomly resampled before each training episode. This means that the AE is retrained from scratch at every training episode. This effectively removes any memory from previous iterations from the AE.

Thanks to the first variant, it is possible to analyze if a random representation is enough to push for exploration and how important is the autonomous learning of  $\mathcal{B}$ . The last variant allows studying the importance of the curriculum effect given by the continuous training of the AE versus the one provided by the data collected in the archive. Note that the only change among all these versions of STAX is the AE training regime. The behavior descriptor is still generated as described in Sec. 5.1. The coverage results for the 3 tested environments are shown in Fig. 9, while the rewards reached in each reward area are shown in Fig. 10.

Not surprisingly, the results show that training the AE, rather than using a randomly generated one, greatly helps the exploration process. The random representations are not enough to discover all the areas of the ground-truth  $\mathcal{B}$ : STAX-NT has significantly lower coverage on all

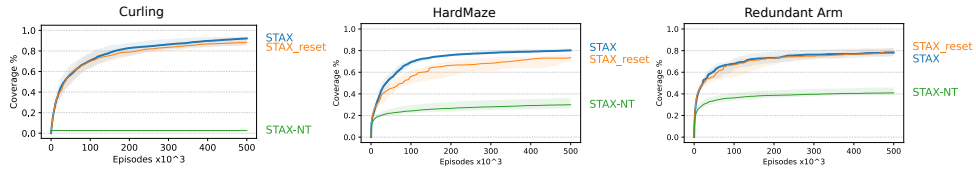


Figure 9: Median coverage with respect to the given evaluation budget reached by STAX against the other versions of the algorithm. The shaded areas represent the 10 and 90 percentile calculated over 15 seeds.

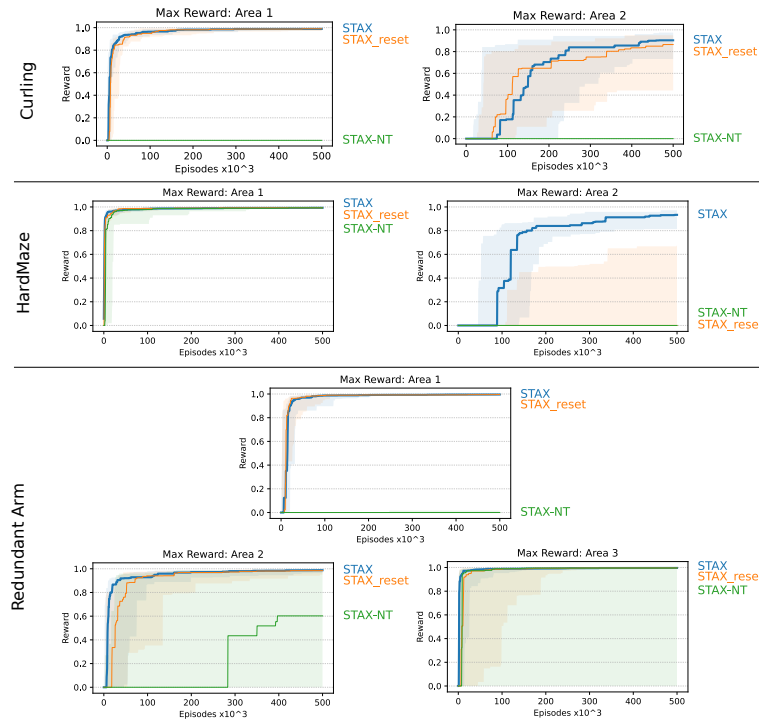


Figure 10: Median maximum reward reached in all the reward areas by STAX against the other versions of the algorithm. The shaded areas represent the 10 and 90 percentile calculated over 15 seeds.

the tested environments compared to the versions in which the AE is trained ( $p < 3.5 \times 10^{-6}$ ). The effect is extreme in the Curling environment in which, to obtain good exploration, it is not enough to randomly move the arm, but it is necessary to properly hit the ball. In the HardMaze and the Redundant Arm environments, the non-trained versions can explore the easier-to-reach areas of the space, but not reach high levels of coverage.

This is reflected in the reward obtained by the methods, shown in Fig. 10. In Curling, random exploration is not enough to discover rewards due to the complex interaction between the ball and the arm, leading to STAX-NT not being able to obtain rewards. At the same time, the random representation suffices to explore just enough to discover the easy-to-reach reward areas in the easier dynamics of the HardMaze and Redundant Arm, allowing the emitters to exploit them.

On the contrary, the continuous training of the AE has a small effect on the coverage: STAX performs similarly to STAX\_reset on both Redundant Arm ( $p = 0.27$ ) and only slightly better on Curling ( $p = 0.038$ ) and HardMaze ( $p = 8.18 \times 10^{-6}$ ). This means that the archive

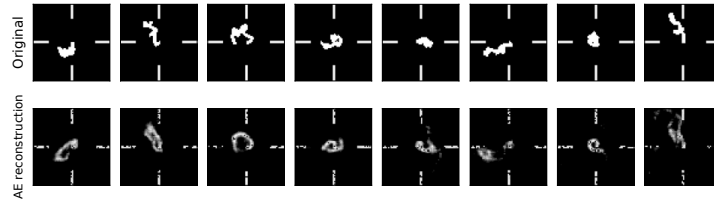


Figure 11: Reconstruction of the AE trained during the search performed by STAX. The first row shows the original  $64 \times 64 \times 3$  images. The second row shows the reconstructions of the images produced by the AE.

can provide enough of a curriculum when learning a representation of  $\mathcal{B}$ . The rewards obtained by the two methods are also similar for all environments on all reward areas, with the exception of the harder-to-reach reward area in the HardMaze environment, for which STAX reaches much higher rewards than STAX\_reset ( $p = 4.49 \times 10^{-5}$ ). The high difference in reward here is due to the fact that this reward area is in the farthest zone from the starting position of the robot. This means that the small difference in exploration between the two methods often prevents STAX\_reset to discover it and thus to exploit it.

## 5.6 Learned behavior space

This section studies how well the trained AE can represent the behavior space and how close the learned representation is to the ground truth one. In Fig. 11 are shown some  $64 \times 64$  observations collected during the evaluation of policies on the Redundant Arm environment (top row) with the respective AE reconstructions (bottom row). This environment provides the hardest to reconstruct observations, given the presence of the whole arm in the images. It is possible to see from the figure that the reconstructed image is not perfect, even though the position of the arm is clear. Nonetheless, this level of reconstruction seems to be enough to push for good exploration in the environment, as seen in Sec. 5.1.

To give a quantitative estimate of the similarity between the learned representation and the ground truth one, we calculated the similarity between the correlation matrices of the ground truth behavior descriptors and the learned descriptors of the policies in the final collections. This is done through the following formula (Herdin et al., 2005):

$$s(C_1, C_2) = 1 - \frac{\text{tr}(C_1 \cdot C_2)}{\|C_1\| \cdot \|C_2\|}, \quad (8)$$

where  $C_1$  and  $C_2$  are the two correlation matrices and the norm is the Frobenius norm. The metric varies between  $[0,1]$  and allows estimating how meaningful is a representation compared to the other. The higher the value of the metric, the closer we can consider the two representations. For comparison, we also evaluate  $s(\cdot, \cdot)$  between the correlation matrix of the ground truth descriptor and the representation given by a random AE over the same observations. The results are shown in Fig. 12.

It is possible to see how representation learned by STAX reaches high values of similarity compared to the ground truth descriptor on all environments ( $p < 4.23 \times 10^{-5}$ ). Moreover, the figure also shows that the representation provided by a random AE is much less meaningful with respect to the original representation, confirming the results from Sec. 5.5.

Finally, Fig. 13 shows an example of how the learned representation for one single reward area can contain multiple distinct zones even if the ground-truth descriptor only contains one, as discussed in Sec. 3.3. This is due to the combined effect of the representation learning done by the AE and the stacking of multiple frames along the trajectory. The presence of multiple reward zones in the learned representations supports the use of emitters when optimizing rewards.



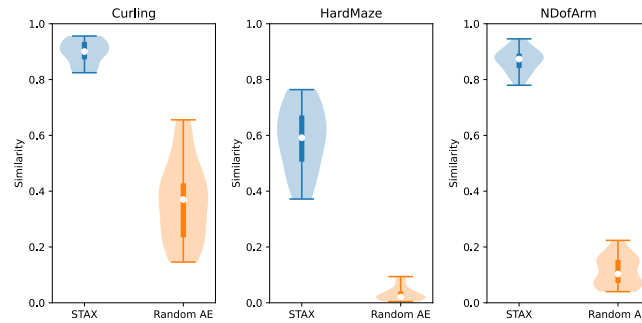


Figure 12: Similarity between the ground truth behavior descriptor and the one provided by the AE for the three environments. The violins are calculated over 15 seeds.

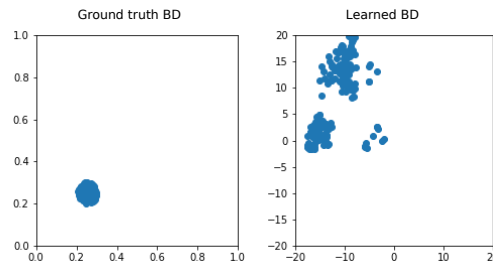


Figure 13: Ground truth descriptor for the policies in reward area 0 (left) compared to the first 2 principal components of the learned descriptor for the same policies (right). The learned descriptor is divided into multiple areas due to the stacking of frames and the learning of the representation by the AE.

## 6 Discussion and Conclusion

This paper introduced STAX, a method that combines the representation learning ability of TAXONS (Paolo et al., 2020) when dealing with unknown  $\mathcal{B}$  and the capacity to focus on interesting areas of the search space of SERENE (Paolo et al., 2021) through emitters. In addition to what TAXONS does when learning  $\mathcal{B}$ , STAX uses multiple observations sampled along the trajectory generated by the policies to extract their behavior descriptor. This allows overcoming the requirement of the final observation needing to be descriptive enough to distinguish between the policies. Moreover, by using a multi-objective approach to combine the two metrics of novelty and surprise, STAX can perform better exploration compared to TAXONS. As discussed in Sec. 3.3, performing reward exploitation through emitters can prove extremely useful when exploring with a learned behavior space. This is due to the fact that there is no guarantee that this learned space will represent all the rewards in a single connected area, as shown in Sec. 5.6.

The results on three different sparse rewards environments show how STAX can prove effective in dealing with these kinds of situations, reaching high performances both from the point of view of exploration and exploitation of the rewards. These results are comparable to the ones obtained by SERENE (Paolo et al., 2021) notwithstanding STAX being provided much less prior information about the task to solve. Moreover, learning the behavior space while performing the search allows reducing the main limitation of NS-based methods: the hand-design of  $\mathcal{B}$ .

It is to notice that, while the choice of learning the behavior space from images might seem limiting, this is not the case. Thanks to the simplicity and availability of cameras, many problems in robotics can be represented through images without providing problem specific information. At the same time, images are only one type of high-dimensional observations and,

while we have not tested STAX on other kind of representation, there is no constraint on the type of observations to use. Finally, while learning the behavior space representation through an AE introduces the need of the model design, this requires much less engineering effort than the one required to hand-design the behavior space. This greatly increases the generalization and applicability of STAX. An example of this is the fact that to solve the three test environment we used the same AE model structure, even if the ground truth behavior space was different.

To properly study how the aspects of policy selection and behavior space learning of STAX influence the exploration process, and the discovery and exploitation of rewards, multiple ablation experiments have been performed. The results show that the combination of using multiple observations collected during the trajectory and the multi-objective policy selection strategy are important in obtaining good coverage of the ground-truth search space. At the same time, the continuous training of the AE during the whole search is shown to provide a negligible curriculum effect, compared to the one provided by training on the data from the archives. Finally, Sec. 5.6 showed how the learned space has a similar structure to the ground truth one, allowing the algorithm to perform good exploration in both.

The introduction of STAX addresses the multiple shortcomings of the original NS algorithm while at the same time opening multiple interesting avenues of research. As for SERENE, STAX uses a simple scheduler to alternate between the exploration and the exploitation processes. Applying more complex and adaptive approaches to perform the switch between the two processes can be an interesting line of work in improving the method even more. Another possible direction of research is the one initiated by Cully (2021), where multiple kinds of emitters are combined through a multi-armed bandit approach. Moreover, the sampling of multiple observations along the trajectory to generate the behavior descriptor leads to interesting questions on how this sampling can be done and how the generated behaviors can be compared more meaningful ways. Recent work started to investigate similar questions (Stork et al., 2020; Hagg et al., 2019) and future work will investigate how an approach like STAX can take advantage of such ideas.

## Acknowledgements

This work has received funding from the European Commission’s Horizon Europe Framework Program under grant agreement No. 101070381 (PILLAR-robots project).

## References

- Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. (2017). Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058.
- Aubret, A., Matignon, L., and Hassas, S. (2019). A survey on intrinsic motivation in reinforcement learning. *arXiv preprint arXiv:1908.06976*.
- Baranes, A. and Oudeyer, P.-Y. (2013). Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73.
- Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, volume 29, pages 1471–1479.
- Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.
- Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018). Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*.

- Cideron, G., Pierrot, T., Perrin, N., Beguir, K., and Sigaud, O. (2020). Qd-rl: Efficient mixing of quality and diversity in reinforcement learning. *arXiv preprint arXiv:2006.08505*.
- Colas, C., Sigaud, O., and Oudeyer, P.-Y. (2018). Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms. In *International Conference on Machine Learning*, pages 1039–1048. PMLR.
- Cully, A. (2019). Autonomous skill discovery with quality-diversity and unsupervised descriptors. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 81–89.
- Cully, A. (2021). Multi-emitter map-elites: improving quality, diversity and data efficiency with heterogeneous sets of emitters. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 84–92.
- Cully, A., Clune, J., Tarapore, D., and Mouret, J.-B. (2015). Robots that can adapt like animals. *Nature*, 521(7553):503.
- Cully, A. and Demiris, Y. (2017). Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22(2):245–259.
- Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197.
- Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. (2019). Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*.
- Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. (2018). Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*.
- Fontaine, M. C., Togelius, J., Nikolaidis, S., and Hoover, A. K. (2020). Covariance matrix adaptation for the rapid illumination of behavior space. In *Proceedings of the 2020 genetic and evolutionary computation conference*, pages 94–102.
- Forestier, S., Portelas, R., Mollard, Y., and Oudeyer, P.-Y. (2022). Intrinsically motivated goal exploration processes with automatic curriculum learning. *J. Mach. Learn. Res.*
- Gaier, A., Asteroth, A., and Mouret, J.-B. (2019). Are quality diversity algorithms better at generating stepping stones than objective-based search? In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 115–116.
- Grillotti, L. and Cully, A. (2022). Unsupervised behaviour discovery with quality-diversity optimisation. *IEEE Transactions on Evolutionary Computation*.
- Hagg, A., Preuss, M., Asteroth, A., and Bäck, T. (2020). An analysis of phenotypic diversity in multi-solution optimization. In *International Conference on Bioinspired Methods and Their Applications*, pages 43–55. Springer.
- Hagg, A., Zaefferer, M., Stork, J., and Gaier, A. (2019). Prediction of neural network performance by phenotypic modeling. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1576–1582.
- Hansen, N. (2016). The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*.
- Herdin, M., Czink, N., Ozcelik, H., and Bonek, E. (2005). Correlation matrix distance, a meaningful measure for evaluation of non-stationary mimo channels. In *2005 IEEE 61st Vehicular Technology Conference*, volume 1, pages 136–140. IEEE.
- Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70.
- Hu, Y., Wang, W., Jia, H., Wang, Y., Chen, Y., Hao, J., Wu, F., and Fan, C. (2020). Learning to utilize shaping rewards: A new approach of reward shaping. *Advances in Neural Information Processing Systems*, 33:15931–15941.

- Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. (2017). Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980.
- Laversanne-Finot, A., Pere, A., and Oudeyer, P.-Y. (2018). Curiosity driven exploration of learned disentangled goal spaces. In *Conference on Robot Learning*, pages 487–504. PMLR.
- Lehman, J. and Stanley, K. O. (2008). Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336.
- Lehman, J. and Stanley, K. O. (2011). Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 211–218. ACM.
- Liapis, A., Martínez, H. P., Togelius, J., and Yannakakis, G. N. (2013). Transforming exploratory creativity with delenox,. In *ICCC*, pages 56–63.
- Loviken, P. and Hemion, N. (2017). Online-learning and planning in high dimensions with finite element goal babbling. In *2017 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 247–254. IEEE.
- Mann, H. B. and Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60.
- Mataric, M. J. (1994). Reward functions for accelerated learning. In *Machine learning proceedings 1994*, pages 181–189. Elsevier.
- Mouret, J.-B. and Clune, J. (2015). Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*.
- Nair, A. V., Pong, V., Dalal, M., Bahl, S., Lin, S., and Levine, S. (2018). Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, pages 9191–9200.
- Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287.
- Oudeyer, P.-Y. and Kaplan, F. (2009). What is intrinsic motivation? a typology of computational approaches. *Frontiers in neurorobotics*, 1:6.
- Paolo, G. (2020). Billiard. <https://github.com/GPaolo/Billiard>.
- Paolo, G., Coninx, A., Doncieux, S., and Laflaquière, A. (2021). Sparse reward exploration via novelty search and emitters. In *The Genetic and Evolutionary Computation Conference 2021 (GECCO 2021)*.
- Paolo, G., Laflaquiere, A., Coninx, A., and Doncieux, S. (2020). Unsupervised learning and exploration of reachable outcome space. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2379–2385. IEEE.
- Pugh, J. K., Soros, L. B., and Stanley, K. O. (2016). Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40.
- Salehi, A., Coninx, A., and Doncieux, S. (2021). Br-ns: an archive-less approach to novelty search. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 172–179.
- Sigaud, O. (2022). Combining evolution and deep reinforcement learning for policy search: a survey. *arXiv preprint arXiv:2203.14009*.
- Stork, J., Zaefferer, M., Bartz-Beielstein, T., and Eiben, A. (2020). Understanding the behavior of reinforcement learning agents. In *International Conference on Bioinspired Methods and Their Applications*, pages 148–160. Springer.

G. Paolo, M. Coninx, A. Laflaquiere, and S. Doncieux

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Trott, A., Zheng, S., Xiong, C., and Socher, R. (2019). Keeping your distance: Solving sparse reward tasks using self-balancing shaped rewards. In *Advances in Neural Information Processing Systems*, pages 10376–10386.