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

Alexandre Coninx alexandre.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. Efficient exploration algorithms have been proposed that require to define a behavior space, that associates to an agent its resulting behavior in a space that is known to be worth exploring. The need to define this space is a 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 efficiently 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-steps 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 are used to 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.

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 behind this 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 that the agent can know how good each performed action is (Sutton and Barto, 2018). These kind of

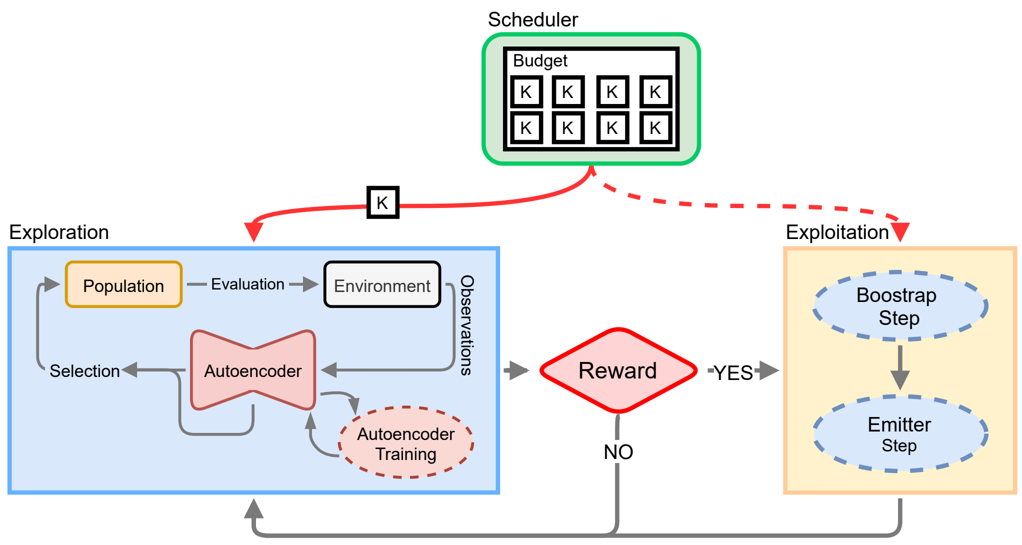

Figure 1: STAX consists of an exploration and an exploitation process alternating thanks to a scheduler. During exploration, the algorithm explores and learns a representation of the behavior space through an AE trained online. Any discovered reward is then exploited in the exploitation step through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

rewards are called *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 performances 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 a 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 and Evolutionary Algorithms (EAs) (Andrychowicz et al., 2017; Trott et al., 2019; Ecoffet et al., 2019; Paolo et al., 2021; Hare, 2019; Riedmiller et al., 2018; Eysenbach et al., 2018; Lehman and Stanley, 2008). 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 an hand-designed space, the behavior space (BS), 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, is often limited by the need to hand-design the BS. While this allows the designer to define what aspects of the problem needs 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 the BS through *representation learning approaches*, thus reducing the amount of prior information needed for the design of the BS itself (Liapis et al., 2013; Paolo et al., 2020; Grillotti and Cully, 2021). 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 BS (Grillotti and Cully, 2021).

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 in instances of reward-based EAs used to perform local search in an area of the whole BS. The idea of emitters was used in SparsE Reward Exploration via Novelty search and Emitters (SERENE) (Paolo et al., 2021) optimize any reward discovered during the search performed by NS. At the same time, SERENE still relies on an hand-designed BS in which to perform the search. STAX builds on SERENE by removing this requirement thanks to the ideas introduced by Paolo et al. (2020) with the Task Agnostic eXploration of Outcome spaces through Novelty and Surprise (TAXONS) algorithm. This approach uses an autoencoder (AE) to learn the behavior space online while performing the search with minimal prior information from the part of the designer.

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-steps 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 what SERENE (Paolo et al., 2021) does.

As a recap, STAX performs three main tasks: (1) it learns a behavior space while (2) exploring it and (3) once a reward is found it exploits it in an efficient way. The method builds on NS by adding an AE (Hinton and Zemel (1994)) to learn a low dimensional representation of the search space, similar to what both Grillotti and Cully (2021) and Paolo et al. (2020) have done. Moreover, the reward is exploited through emitters, a concept introduced by Fontaine et al. (2020) and used by Cully (2020) and Paolo et al. (2021) to quickly improve on rewards. The advantages provided by STAX are twofold: (1) it can efficiently deal with sparse rewards situations, 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 BS, it removes the limitation of classical divergent-search approaches requiring an hand-designed search space, thus greatly reducing the amount of prior informations needed at design time. All of this allows STAX to efficiently 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 the goal the policy should solve is (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 situation 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 in augmenting the original reward function with additional features that are supposed to provide the agent with a 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 of 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., 2017; Laversanne-Finot et al., 2018). Forestier et al. (2017) use this to firstly learn a goal-parametrized policy and then use this policy to solve the given task. These approaches have also been used with two-phases 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), that 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 Quality-Diversity (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

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 on 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 θ_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:

$$\phi(\theta_i) = b_i. \tag{1}$$

Once the behavior descriptors of all the policies in a population have been calculated, the novelty of a policy θ_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:

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

where J is the set of indexes of the k closest policies to θ_i in the outcome space.

At each generation, the novelty of the policies is calculated and the ones with 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 an outcome descriptor

At the core of many divergent search algorithms lies a hand-designed BS. The need to hand-design this space poses strong limitations for the application of these methods to various problems in which it is not clear what are the factors important for the exploration. 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, 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*. 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 the BS. 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 most innovative policies.

2.4 Emitters

Notwithstanding its exploration capabilities, vanilla NS is not equipped to take advantage of any reward that can be found during the search. This limits the power of the algorithm and discards some important information on the task that can be used to steer and improve the efficiency of the search. Many solutions have been proposed to address this problem (Lehman and Stanley (2011); Mouret and Clune (2015); Cully et al. (2015); Cully (2020); Paolo et al. (2021)), leading to the development of the QD family of algorithms (Pugh et al. (2016); Cully and Demiris (2017)).

Among QD algorithms worth of notice are approaches using *emitters*. Introduced by Fontaine et al. (2020) and later used by Cully (2020) and Paolo et al. (2021), emitters are instances of reward-based EAs instantiated during the search performed by another EA to quickly explore 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 θ_i from the MAP-Elites archive and using it to initialize the population of an emitter $\mathcal{E}_i$, that 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 θ_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-stages process, one performing exploration, the other exploiting the found rewards. Exploration is done through NS over the hand-designed BS $\mathcal{B}$. 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 allows the algorithm to be more efficient in situations of sparse rewards 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 a BS 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 BS, it is possible to quickly explore an unknown search space while efficiently optimizing any possible discovered reward.

3 Method

STAX deals with the limitations of NS for *sparse rewards* settings by separating the search process in two alternating sub-processes: one performing *exploration* of the search space and another performing *exploitation* of any discovered reward. This allows STAX to find different high reward policies with minimal prior information about the task. The alternation between the two processes is performed through a *meta-scheduler* whose task 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, as in the SERENE algorithm (Paolo et al., 2021). At the same time, STAX reduces the amount of prior information needed to solve the task by removing the need to hand-design the behavior space (BS). 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). The AE is trained online on the data generated by the evaluation of the policies $\theta_i \in \Theta$. The encoder part of the AE can then be used as *observation function* and its *feature space* $\mathcal{F}$ as behavior space $\mathcal{B}$.

The algorithm starts with the *exploration phase*, in which exploration of the behavior space $\mathcal{B}$ is performed. The search is driven by using the information extracted by the AE from

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;
RESULT: Novelty archive $\mathcal{A}_{\text{Nov}}$, rewarding archive $\mathcal{A}_{\text{Rew}}$, trained AE;
$\mathcal{A}_{\text{Nov}} = \emptyset$;
$\mathcal{A}_{\text{Rew}} = \emptyset$;
$\mathcal{Q}_{\text{Em}} = \emptyset$;
$\mathcal{Q}_{\text{Cand\_Nov}} = \emptyset$;
$\mathcal{Q}_{\text{Cand\_Em}} = \emptyset$;
$D = 0$;
Initialized training counter $TI_C = 0$;
Sample population $\Gamma_0$;
Split $Bud$ in chunks of size $K_{Bud}$;
while $Bud$ not depleted do
    if $\Gamma_0$ then
        Evaluate $\theta_i, \ \forall\theta_i \in \Gamma_0$;
        Calculate $b_i = \phi(\theta_i) \in \mathcal{B}, \ \forall\theta_i \in \Gamma_0$;
    Exploration Phase ($K_{Bud}$, $m$, $\sigma$, $\mathcal{A}_{\text{Nov}}$, $\mathcal{Q}_{\text{Cand\_Em}}$, $\Gamma_g$, $Q$, AE);
    $TI_C = TI_C + 1$;
    if $TI_C == TI$ then
        $DS$ = Extract dataset($\mathcal{A}_{\text{Nov}}$, $\mathcal{A}_{\text{Rew}}$, $\Gamma_g$, $\Gamma_g^m$);
        Train Autoencoder (AE, $DS$);
        Update descriptors (AE, $\Gamma_g$, $\Gamma_g^m$, $\mathcal{A}_{\text{Nov}}$, $\mathcal{A}_{\text{Rew}}$, $\mathcal{Q}_{\text{Em}}$, $\mathcal{Q}_{\text{Cand\_Nov}}$, $\mathcal{Q}_{\text{Cand\_Em}}$);
        $TI = TI + 1$;
        $TI_C = 0$;
    if $\mathcal{Q}_{Cand\_Em}! = \emptyset$ or $\mathcal{Q}_{Em}! = \emptyset$ then
        Exploitation Phase ($K_{Bud}$, $\mathcal{Q}_{\text{Cand\_Em}}$, $\lambda$, $m$, $\mathcal{Q}_{\text{Em}}$, $\mathcal{A}_{\text{Nov}}$, $\mathcal{A}_{\text{Rew}}$, $M_{\mathcal{E}}$);
```

the observations collected during the evaluations of the policies. This same observations are used to train the AE online in an unsupervised way after the exploration steps. The training process allows STAX to learn a low-dimensional representation of the BS that is used to drive the search. For the first iterations of the search, the AE representation is still immature, so the training happens more frequently; but once few training iterations have been performed, the AE can better represent the behaviors, so the training happens less and less frequently. Finally, if a policy θ_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 being that behaviors similar to the rewarding behavior $f(\theta_i)$ are likely rewarding too, with possibly even higher performances than $f(\theta_i)$. These behaviors can be considered to belong to the subspace of rewarding behaviors $\mathcal{B}_{Rew} \in \mathcal{B}$ and their corresponding policies can be discovered by performing local search around θ_i through emitters. Note that the reward exploitation performed during this phase does not rely on any behavior descriptor. The quality of the BS 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 the BS has not been learned yet, STAX can still exploit it with great efficiency thanks to descriptor-less emitters.

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

- *novelty archive* $\mathcal{A}_{\text{Nov}}$: a collection of policies with diverse behaviors found during the *exploration phase*. This is one the two collections of policies returned as output of STAX;
- *reward archive* $\mathcal{A}_{\text{Rew}}$: a set of the most rewarding policies found during the *exploitation phase*. This is the other collection of policies returned as output of STAX;
- *candidates emitter buffer* $\mathcal{Q}_{\text{Cand\_Em}}$: a buffer in which the rewarding policies $\phi(\theta_i) \in \mathcal{B}_{\text{Rew}}$ 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.

These sets are the same as the ones used by SERENE, and an high-level overview of their interactions is shown in Fig. 2.

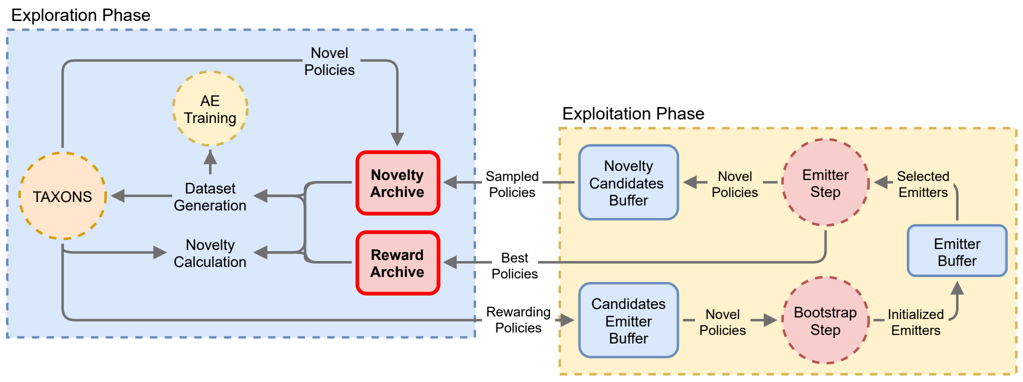

Figure 2: Overview of the containers used during the search by STAX to track the discovered policies and the initialized emitters. The two collections returned as output of the algorithms are highlighted in red.

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 M policies $\theta \in \Theta$ in the initial population Γ_0 are sampled from a normal distribution $\mathcal{N}(0, \mathbb{I})$. The AE used to drive the search is also randomly initialized. [1] At each generation g, m policies θ_i^j are generated for each policy θ_i in the current population Γ_g through a *mutation operator*. This will result in an offspring population Γ_g^m of size $m \times M$ whose policies are formed as:

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

[1]The initialization is done through the default Pytorch initialization.

The policies in the offspring population $\theta \in \Gamma_g^m$ are then evaluated. During the evaluation of a policy θ_i the system traverses a trajectory of states $\tau_s^i = [s_0^i, \dots, s_T^i]$ that are observed through 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 obtained by using *multiple observations* sampled along τ_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 according to Eq. (4):

$$f(\theta_i) = [\dots, E(o_{t_k}^i), \dots, E(o_{t_K}^i)], \tag{4}$$

where $o_{t_k}^i$ is the observation generated by the policy θ_i at time-step t_k. Using multiple observations along the trajectory allows to remove the assumption, done by Paolo et al. (2020), that the last observation contains enough information to describe the whole behavior of a policy.

The diversity of a policy is evaluated through two metrics: *novelty* and *surprise*. The first one is similar to NS novelty from Eq. (2), in which $\phi(\theta_i)$ is represented by $f(\theta_i)$. This can be represented as in Eq. (5):

$$\eta(\theta_i) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(f(\theta_i), f(\theta_j)). \tag{5}$$

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 θ_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:

$$s(\theta_i) = \sum_{k \in K} \left|\left| o_{t_k}^i - D\big(E(o_{t_k}^i)\big) \right|\right|^2, \tag{6}$$

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 Γ_{g+1} through the NSGA-II multi-objective approach (Deb et al., 2002). This is in contrast to what 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, N_Q policies are uniformly sampled 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 in the observations used to generate the behavior descriptor of the policies, as defined in Eq. (4). 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 Γ_g and the offspring population Γ_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.

Algorithm 2: STAX Exploration Phase

```
INPUT: budget chunk $K_{Bud}$, number of offspring per parent $m$, mutation parameter
  $\sigma$, novelty archive $\mathcal{A}_{\text{Nov}}$, candidate emitters buffer $\mathcal{Q}_{\text{Cand\_Em}}$, population $\Gamma_g$, number
  of policies $N_Q$, autoencoder AE;
while $K_{Bud}$ not depleted do
    Generate offspring $\Gamma^m_g$ from population $\Gamma_g$;
    Evaluate $\theta_i, \ \forall \theta_i \in \Gamma^m_g$;
    Calculate $b_i = \phi(\theta_i) = [\dots, E(o^i_{t_k}), \dots, E(o^i_{t_K})] \ \forall \theta_i \in \Gamma^m_g$;
    Calculate $\eta(\theta_i) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(b_i, b_j), \ \forall \theta_i \in \Gamma^m_g$;
    Calculate $s(\theta_i) = \sum_{k \in K} \left|\left| o^i_{t_k} - D\big(E(o^i_{t_k})\big) \right|\right|^2 \ \forall \theta_i \in \Gamma^m_g$;
    $\mathcal{A}_{\text{Nov}} \leftarrow N_Q$ samples from $\Gamma^m_g$;
    if $\phi(\theta_i) \in \mathcal{B}_{Rew}$ then
        $\mathcal{Q}_{\text{Cand\_Em}} \leftarrow \theta_i$
    /* NSGA-II based policy selection */
    Calculate non dominated fronts $F_j, \ \forall \theta_i \in \Gamma^m_g \bigcup \Gamma_g$;
    Sort fronts according to non domination;
    Generate $\Gamma_{g+1}$ from most non dominated solutions $\theta_i \in F_j$;
    if 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 $\theta_i \in F_J$;
```

Once the dataset DS has been collected, it is split into two sub-datasets: the *training dataset* DS_{Train} and the *validation dataset* DS_{Val}. For each training episode, the AE is trained on the DS_{Train}. At the end of each training epoch on DS_{Train}, the model validation error is calculated on D_{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 employed in the AURORA method (Cully (2019)). This allows to adapt the frequency of the training to the maturity of the learned BS, while saving time and computational resources with respect to training the AE at fixed intervals. Moreover, by training less frequently, the possible overfitting of the AE on the data present in the archives is limited. 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 allows to keep 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_{Bud} to the exploitation step. The objective of this phase is to evaluate the emitters to improve on the reward. This is done through two sub-steps: the *bootstrap step* and the *emitter step*. During the bootstrap step, the policies in the *emitters candidate buffer* $\mathcal{Q}_{\text{Cand\_Em}}$ are used to initialize emitters that are quickly evaluated to find the ones with potential to improve on the reward. The emitters with such potential are then added to the *emitter buffer* $\mathcal{Q}_{\text{Em}}$ to be fully evaluated during the emitter step. At the same time, the emitters not capable of improving on the reward are discarded. This ensures that only promising emitters are considered for full evaluation in the emitter step, thus

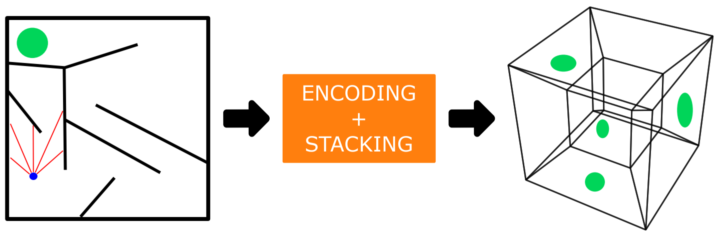

Figure 3: The behavior space generated by stacking the learned representations from multiple observations generated during the search can contain multiple reward areas. Even if the original ground-truth space contained only one.

preventing a waste of evaluation budget.

Using emitters allows to disjointly optimize multiple reward areas in an efficient way. This is fundamental for an approach like STAX in which the BS is autonomously learned. In hand-designed BS the engineer has total control over the BS 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 settings there is no guarantee that the new BS will have the same structure of the reward areas as the ground-truth hand-designed BS. Given the complex nature of the learned BS, 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 BS, as shown in Fig. 3. The effect is even more likely in the first phases of the search, when the AE is not yet properly trained and its feature space not completely mature. 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 BS representation is so complex.

In the following we will describe in detail how the two sub-steps of the exploitation process work and how the reward is optimized.

Bootstrap step

The candidates 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 $\mathcal{B}_{\text{Rew}}$.

An emitter is an instance of a *reward-based EA*. In this paper we use as emitters 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 it 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_γ of size $M_\mathcal{E}$ of policies $\tilde{\theta}_i \in \Theta$, its offspring population P^m_γ 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 γ, and a tracker for the highest reward found until now R_γ. At the same time, the emitter also tracks two novelties, η_γ, that is the novelty of the most novel policy found until generation γ, and the

Algorithm 3: STAX Exploitation Phase

```
INPUT: budget chunk $K_{Bud}$, candidate emitters buffer $\mathcal{Q}_{\text{Cand\_Em}}$, number of bootstrap
  generations $\lambda$, emitter population size $M_{\mathcal{E}}$, number of offspring per policy $m$,
  emitters buffer $\mathcal{Q}_{\text{Em}}$, rewarding archive $\mathcal{A}_{\text{Rew}}$, novelty archive $\mathcal{A}_{\text{Nov}}$;
/* Bootstrap step */
while $K_{Bud}/3$ not depleted do
    Select most novel policy $\theta_i$ from $\mathcal{Q}_{\text{Cand\_Em}}$;
    Calculate $\sigma_i$;
    Initialize: $\mathcal{E}_i$, $\mathcal{Q}^i_{\text{Cand\_Nov}} = \emptyset$, and $P_0$;
    for $\gamma \in \{0, \dots, \lambda\}$ do
        if $P_0$ then
            Evaluate $\tilde{\theta}_j, \forall \tilde{\theta}_j \in P_0$;
        Generate offspring population $P^m_\gamma$ from $P_\gamma$;
        Evaluate $\tilde{\theta}_j, \forall \tilde{\theta}_j \in P^m_\gamma$;
        Generate $P_{\gamma+1}$ from best $\tilde{\theta}_j \in P^m_\gamma \bigcup P_\gamma$;
    Calculate $I(\mathcal{E}_i)$;
    if $I(\mathcal{E}_i) > 0$ then
        $\mathcal{Q}_{\text{Em}} \leftarrow \mathcal{E}_i$;
/* Emitters step */
Calculate pareto fronts in $\mathcal{Q}_{\text{Em}}$;
while $2/3 K_{Bud}$ not depleted do
    Sample $\mathcal{E}_i$ from non-dominated emitters in $\mathcal{Q}_{\text{Em}}$;
    while not $terminate(\mathcal{E}_i)$ do
        Generate offspring population $P^m_\gamma$ from $P_\gamma$;
        Evaluate $\tilde{\theta}_j, \forall \tilde{\theta}_j \in P^m_\gamma$;
        $\mathcal{A}_{\text{Rew}} \leftarrow \tilde{\theta}_j, \ \forall \tilde{\theta}_j \in P^m_\gamma \mid r(\tilde{\theta}_j) > R_\gamma$;
        $\mathcal{Q}^i_{\text{Cand\_Nov}} \leftarrow \tilde{\theta}_j, \ \forall \tilde{\theta}_j \in P^m_g \mid \eta(\tilde{\theta}_j) > \eta_i$;
        Generate $P_{\gamma+1}$ from best $\tilde{\theta}_j \in P^m_\gamma \bigcup P_\gamma$;
        Update $I(\mathcal{E}_i)$ and $R_\gamma$;
        if $terminate(\mathcal{E}_i)$ then
            $\mathcal{A}_{\text{Nov}} \leftarrow N_Q$ samples from $\mathcal{Q}^i_{\text{Cand\_Nov}}$;
            Discard emitter $\mathcal{E}_i$;
```

emitter novelty, $\eta(\mathcal{E}_i)$, corresponding to the novelty of the policy generating the emitter. The emitter is initialized from the policy θ_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 θ_i and the closest θ_j corresponds to 3 standard deviations. This is done by initializing σ_i as:

$$\sigma_i = \frac{\min_j(\text{dist}(\theta_i, \theta_j))}{3}, \ \ \forall \tilde{\theta}_j \in \Gamma^m_g \cup \tilde{\Gamma}_g. \tag{7}$$

Once an emitter $\mathcal{E}_i$ has been initialized, it is executed for λ generations to evaluate its potential for improving the reward. This potential is expressed through the *emitter improvement* $I(\mathcal{E}_i)$, calculated as the difference between the average reward obtained during the most recent and the initial generations of the emitter. 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 reward are low, so it is not worth to allocate more evaluation budget to it. For this reason only the emitters with positive improvement are added to the *emitter buffer* $\mathcal{Q}_{Em}$ further evaluation during the emitter step, while the rest are discarded. The improvement is expressed as:

$$I(\mathcal{E}_i) = \frac{1}{\lambda M_{\mathcal{E}}} \left(\sum_{\gamma=T-\lambda/2}^{T} \sum_{j=0}^{M_{\mathcal{E}}} r_{(\gamma,j)} - \sum_{\gamma=\gamma_0}^{\lambda/2} \sum_{j=0}^{M_{\mathcal{E}}} r_{(\gamma,j)} \right), \tag{8}$$

where T is the last evaluated generation, $r(\gamma, j)$ is the reward of policy $\tilde{\theta}_j \in P_\gamma$ and γ_0 is the generation at which the emitter is at the beginning of its evaluation. The whole bootstrap step lasts $K_{bud}/3$ 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* $\mathcal{Q}_{Em}$. 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 γ, 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 happen: the $2/3 K_{Bud}$ 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 $\mathcal{Q}_{Em}$ 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.

There can be many *termination criteria*, depending on the kind of algorithm used as emitter. In this work we use the termination criterion 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). The emitter $\mathcal{E}_i$ is stopped if it cannot improve anymore on the reward. This is calculated by tracking the history of the rewards obtained by the emitter over the last $120 + 20 * n \backslash M_{\mathcal{E}}$, where n is the size of the parameter space Θ 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.

The whole exploitation phase is detailed in Algorithm 3. The code repository is available at: `<url hidden for review process>`.

4 Experiments

This section studies how STAX can discover highly rewarding policies while exploring an outcome space learned on the fly. All of this with minimal previous information about the environment and the task at hand. STAX will be compared against various baselines. Moreover, multiple ablation studies will be performed to study which aspects of the method are the most important ones. In order to perform this analysis, STAX is evaluated on 3 sparse rewards environments, shown in Fig. 4.

Curling: it consists of a 2 Degrees of Freedom (DoF) arm pushing a ball over a table (Paolo et al., 2021). The arm is controller by a 3 layers Neural Network (NN) with each layer of size 5.

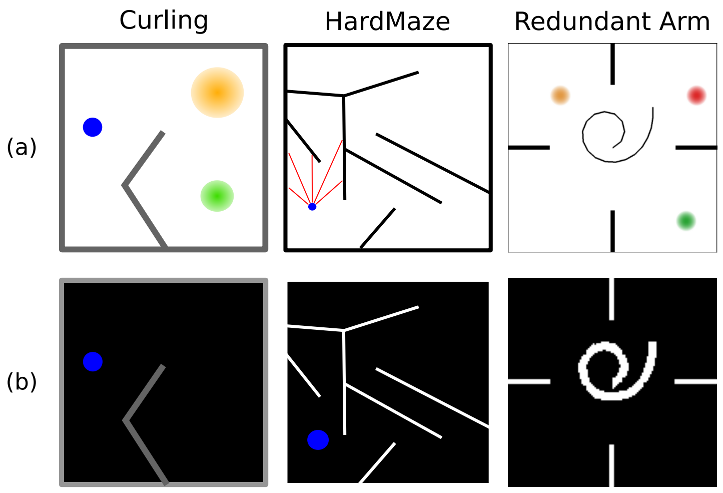

Figure 4: The three testing environments. Row (a) shows the real environments with the reward areas. Row (b) contains the 64×64 RGB observation of the environment as seen by the AE. The behavior descriptors are generated by sampling 5 of these images along the trajectories.

The input of the controller is a 6-dimensional array containing the (x, y) ball pose and the two 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 BS representation is the final (x, y) position of the ball. The environment, together with the 64×64 RGB image the AE sees during the algorithm execution, is shown in Fig. 4.

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. 4, 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. 4, 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 to the center 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 BS representation is the final (x, y) position of the robot. The environment, together with the 64×64 RGB image the AE sees during the algorithm execution, is shown in Fig. 4.

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 joints' positions, while the output consists in the 20-dimensional joints' torque vector. The policies are run for 100 timestep 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 BS 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 64×64 RGB image the AE sees during the algorithm execution, is shown in Fig. 4.

In all of 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 50×50 grid to cover the ground-truth behavior space of every 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.

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 $M_{\mathcal{E}} = 6$ 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 AE is trained with the Adam optimizer (Kingma and Ba, 2014) with an learning rate of 0.001. The statistical results are computed over 15 runs for each experiment. Finally, in each plot, the performances of methods using the ground-truth BS are represented with dashed lines, while the methods learning the BS are shown through a continuous line.

5 Results

In this section, the results obtained in the experiments are discussed.

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* BS defined in Sec. 4 for each one of the tested environments. The coverage metric is evaluated by dividing said ground truth space into a 50×50 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

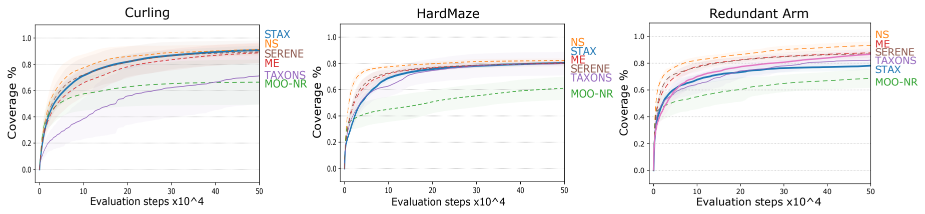

Figure 5: Average coverage with respect to the given evaluation budget reached by STAX against the different baselines. The shaded areas represent one standard deviation.

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 for a single final position of the ball - the one considered in the ground-truth BS - can correspond multiple arm positions that are represented by STAX.

Fig. 5 shows the coverage reached by our method and all the tested baselines. It can be seen that STAX can perform exploration on a level comparable with NS on all the environments, except on the Redundant Arm, in which the coverage is lower. In this environment, STAX learns 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 performance of STAX when the AE is shown only the end effector position, rather than the whole arm (pink line in the Redundant Arm plot in Fig. 5), are comparable to the ones of SERENE. The methods using the hand-designed ground-truth BS to drive the search - ME and SERENE - reach high levels of coverage too. This is expected given that both method perform the search in the same space in which the coverage metric is computed. The good performance of STAX are instead obtained with minimal information about the task and the space in which those would be measured. 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. 4, that can act as a distractors in situations in which only the final position of the ball is interesting. At the same time, the presence of the arm is not an hindrance to the performances of STAX. This is likely be 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 more efficient 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. 6. Using emitters to exploit the reward allows STAX to reach almost the maximum reward in few evaluations. These performances are similar to the ones obtained by SERENE, 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 with much slower pace than STAX. This is not the case for the multi-objective approach MOO-NR, that 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,

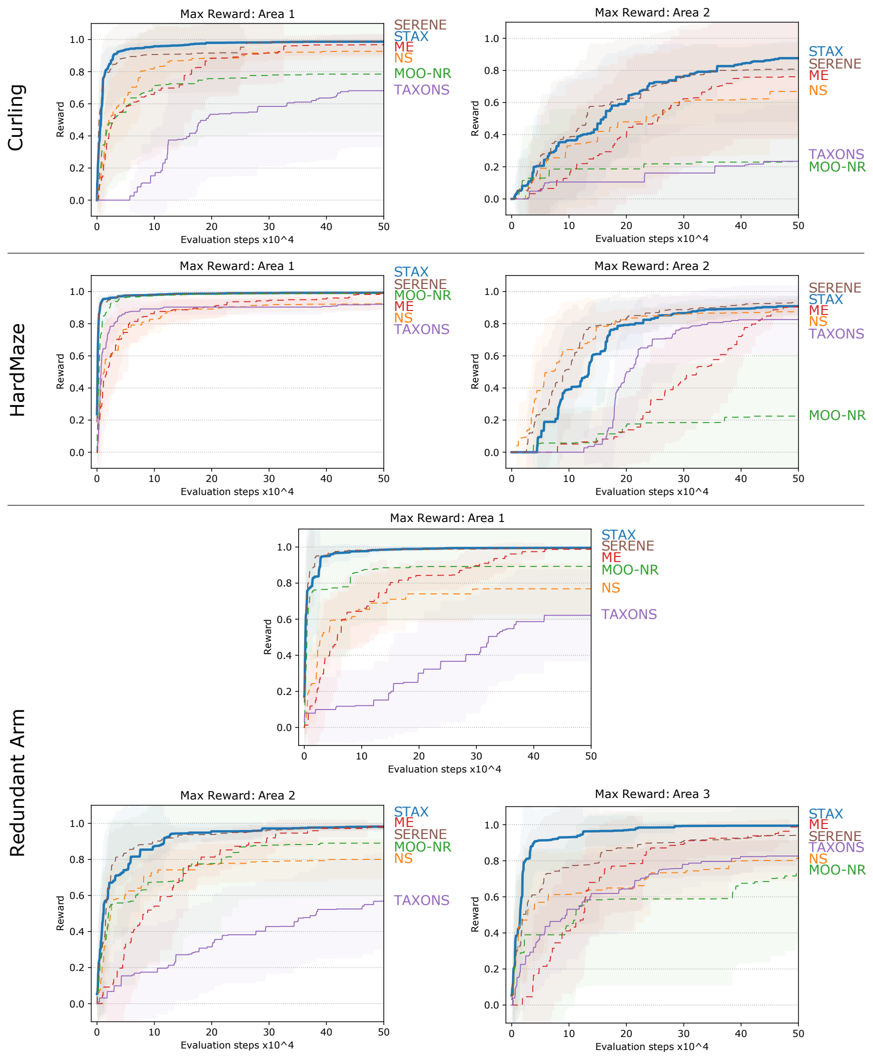

Figure 6: Average maximum reward reached in all the reward areas by STAX against the different baselines. The shaded areas represent one standard deviation.

with TAXONS being consistently worse in this regard. This is due to the lack of any reward-exploitation mechanism present in both methods. This is even more noticeable on the redundant arm environment, where even if TAXONS can reach similar coverage levels than STAX, the absence of any reward improving mechanism leads to very low performances on all reward areas.

5.3 Final archives distribution

The final distribution of the behaviors representations for the policies in the final archives is shown in Fig. 7. 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.

The coverage of the reward areas for STAX and SERENE is similar, both approaches using emitters to exploit the rewards. At the same time, STAX tends to cover the search space less densely than SERENE and NS, due to not knowing the ground-truth BS at search time. The coverage of the space for STAX more closely resembles the one obtained by TAXONS, the other method learning the BS representation at search time, with the exception of the reward areas, better covered by STAX. Similar coverage of the reward areas is obtained by MOO-NR, but this comes at the cost of exploration of the rest of the search space. ME also obtains a uniform distribution over the whole space, thanks to the a priori discretization of the BS.

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 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;
- **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;
- **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 areas are analyzed.

The average coverage is shown in Fig. 8. It is possible to see how the variants using multiple observations of the trajectory perform consistently better on all environments. This is also the case for the Redundant arm environment in which, while the final coverage of the algorithms is equivalent, the two variants using multiple observations tend to reach higher levels quicker. This is due to the AEs of these variants being trained on 5 times more data than the ones of the variants using a single observation.

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. 9. In each of the reward areas of all environments, STAX_multi reaches the highest performances in the quickest fashion. In general the methods using only the last observation to extract a description of the behavior of a policy perform the worst. At the same time, the multi-objective policy selection method, used by both STAX_multi and STAX_single, has a weaker but non negligible effect on both exploration and the exploitation. It can be seen in fact that the version using both multiple observations and the multi-objective policy selection strategy performs consistently better than all the other variants.

5.5 Autoencoder training regime

This section analyzes how the way the BS 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

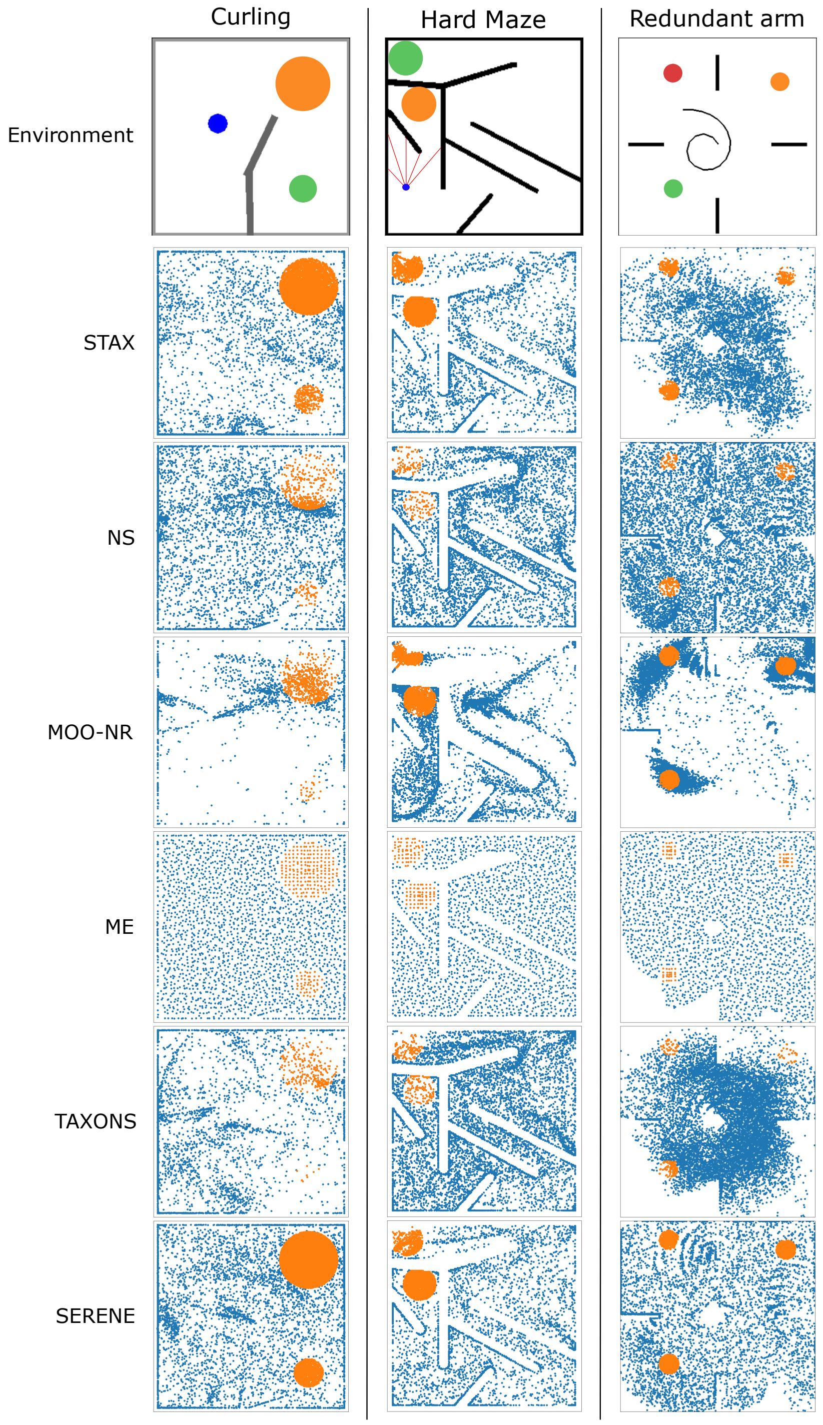

Figure 7: Distribution of the behavior descriptors of the archived policies. On each column are shown the results for an environment, while on each row is shown the distribution for each experiment. The archive plotted are from the runs achieving highest coverage. In blue are the policies with no reward, in orange the policies with a reward. For STAX and SERENE in blue are the policies in the novelty archive and in orange the policies in the reward archive.

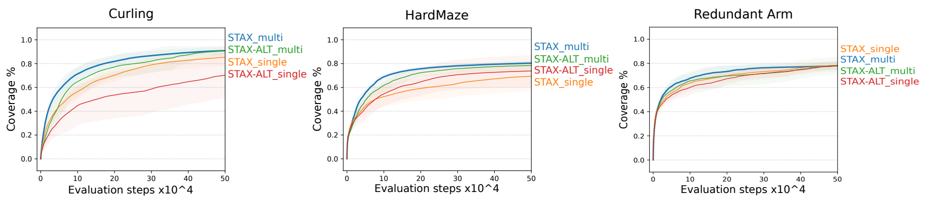

Figure 8: Average coverage with respect to the given evaluation budget reached by STAX against the ablated versions of the algorithm. The shaded areas represent one standard deviation.

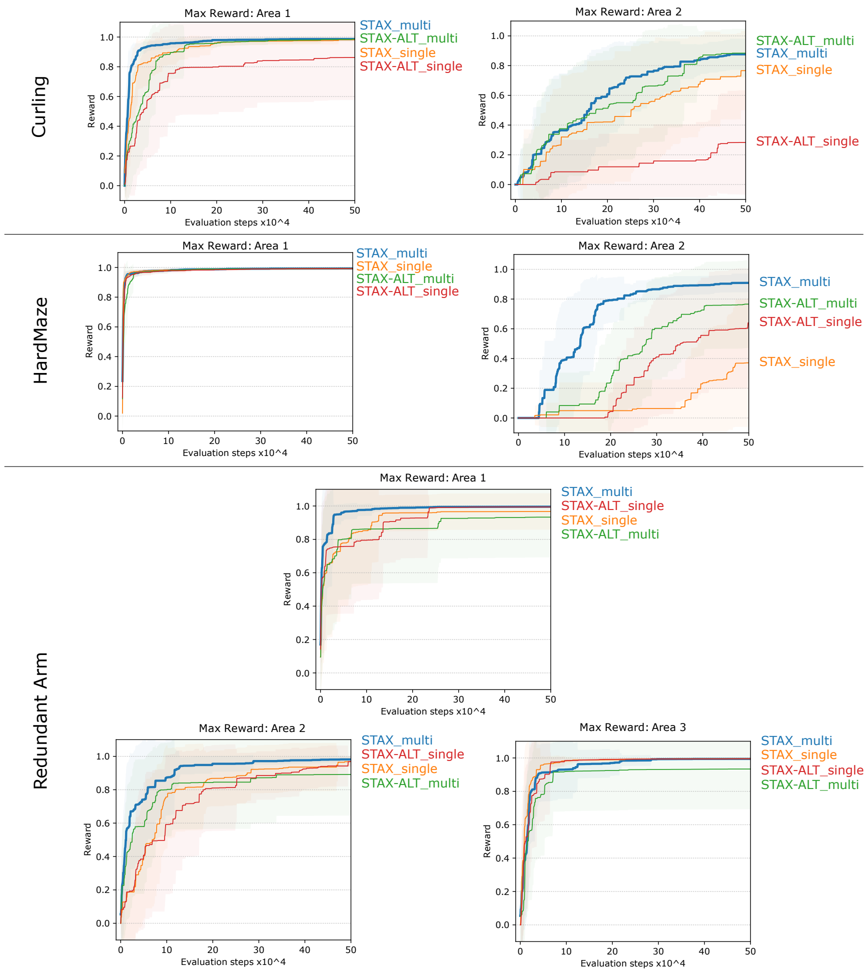

Figure 9: Average maximum reward reached in all the reward areas by STAX against the ablated versions of the algorithm. The shaded areas represent one standard deviation.

any influence on the search process. In STAX the AE is continuously trained across different training episodes. This means that, similarly to what done by Paolo et al. (2020), the training of 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 few elements.

To analyze these two aspects, STAX is compared against 3 variants:

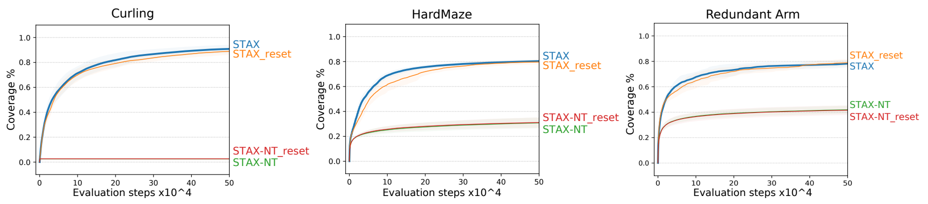

Figure 10: Average coverage with respect to the given evaluation budget reached by STAX against the other versions of the algorithm. The shaded areas represent one standard deviation.

- **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-NT_reset**: in which the search is driven through an AE whose weights are randomly sampled every TI exploration steps. This means that every time the vanilla version of STAX would train the AE, this version randomly samples new weights for the AE;
- **STAX_reset**: in which the weights of the AE are randomly resampled before each training episode. This effectively removes any memory from previous iterations from the AE.

Thanks to the first two variants, it is possible to analyse if a random but constant representation is better than a continuously changing random representation to drive the search. The last variant allows to study the importance of the curriculum effect given by the continuous training of the AE. 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. 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 BS, even if said representations change during the search, as is the case for the STAX-NT_reset variant. At the same time, the continuous training of the AE does not have a big effect on the coverage in any of the environments. This means that the archive can provide enough of a curriculum when learning a representation of the BS. However, the retraining from scratch at every training episode of the AE increases the execution wall-time of STAX_reset compared to STAX.

The results show how the variants in which the BS representation is not learned really struggle to explore a big part of the space. This 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.

These experiments clearly show that each environment has different dynamics when it comes to exploration. This strengthens our assumption that hand-designing a BS in order to properly explore can be difficult and require adaptations to each single situation. For this reason, it is important to design algorithms like STAX that can learn said BS online while starting with minimal prior information. These algorithms should adapt to all environment dynamics by taking advantage as much as possible of the data generated during the search.

5.6 Learned behavior space

This section studies how well the trained AE can represent the BS and how close these learned representations are to the ground-truth one. Given that the results are comparable among all the environments, the section will focus mainly on the harder to explore Redundant Arm environment. Fig. 11 shows how well STAX's learned AE can reconstruct the observations collected during the evaluation of the policies. The first row shows the 64×64 RGB final observation of the trajectories of a set of policies sampled form the final archives. In the second

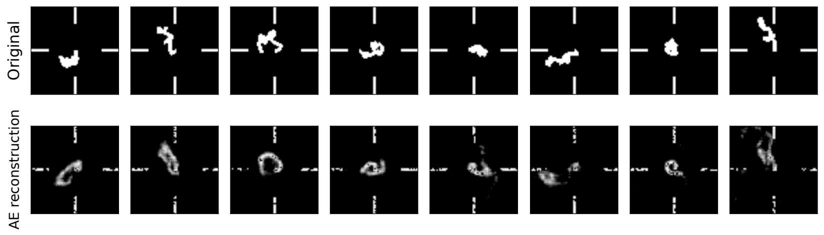

Figure 11: Reconstruction of the AE trained during the search performed by STAX. The first row shows the original $64 \times 64 \times 3$ images. The second row shows the reconstructions of the images produced by the AE.

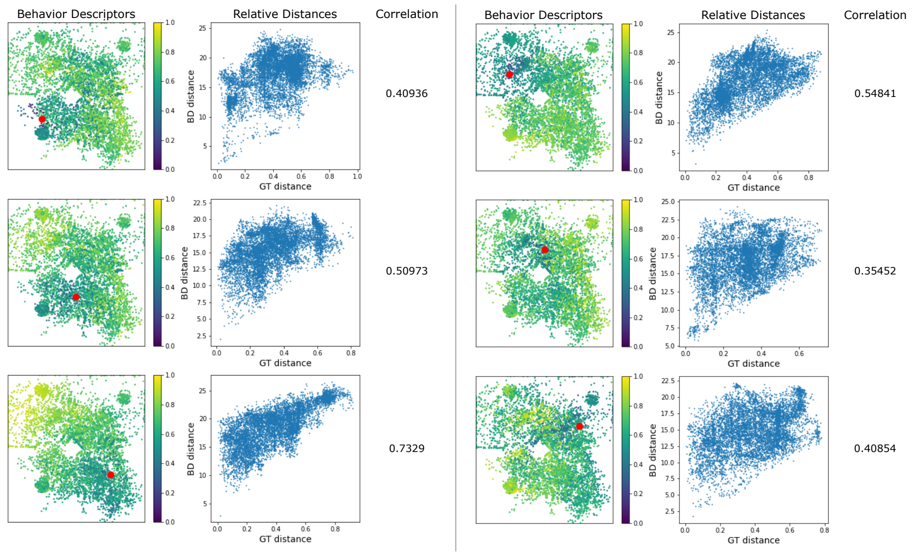

Figure 12: Representation of the proximity between the learned BS and ground truth one. The first column represents the points in the ground truth BS color coded according to the distance in the learned BS. The red circle is the sampled descriptor for which the distance from the other descriptors is calculated. The second column represent the distances in the learned space with respect to the ground truth BS. Finally, on the third column is shown the correlation coefficient calculated between said distances in the two spaces.

row are shown the reconstructions produced by the trained AE. While from this reconstruction it is possible to understand the position of the arm, the image is not perfect. Nonetheless, this level of reconstruction accuracy seems to be enough to push for good exploration in the environment, as seen in Sec. 5.1.

From this observation, the question on how close the learned BS representation is to the ground truth one arises naturally. This has been studied by sampling 6 policies from the archives at 6 different positions in the ground truth BS. Then, the distance between the learned behavior descriptors of the sampled policies and the ones of the other policies in the archives is calculated. The results can be seen in Fig. 12. Each row shows the results with respect to one of the 6 sampled policies, whose ground truth descriptor is highlighted in red in the plot in the first column.

The first column contains the policies' ground-truth descriptors plotted by color coding them according to their distance in the learned BS. Closer points in this space are represented in darker colors, farther ones in lighter colors. The second column represents the distances in the learned BS with respect to the distances in the ground-truth space, while on the third column it is shown the Pearson correlation coefficient between these distances (Pearson, 1895).

From the figure it is possible to see that closer points in the learned BS are closer in the ground-truth (GT) space, and the farther these points are in the GT space, the farther they become in the learned space. This is also shown by analyzing the correlation between the distances in the two spaces. From this, it is possible to see that there is moderate to high correlation between these distances, confirming that closer points in the ground-truth space tend to be closer in the learned space and vice-versa. This means that the AE has learned a meaningful representation that can be used to push for exploration by calculating distances in the learned space, proving the efficacy of this approach.

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 BS 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 the BS, STAX uses multiple observations sampled along the trajectory generated by the policies to extract their behavior descriptor. This allows to overcome 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 BS. This is due to the fact that there is no guarantee that this learned BS will represent all the rewards in a single connected area.

The results on three different sparse rewards environments show how STAX can prove effective in dealing with this kind 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 outcome space while performing the search allows to remove the main limitation of NS-based methods: the hand-design of the BS.

To properly study how the aspects of policy selection and BS 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. Moreover, the continuous training of the AE during the whole search is shown to provide an useful curriculum effect, in addition to the one provided by training on the data from the archives. Finally, Sec. 5.6 showed how the learned BS has a similar structure to the ground truth BS, 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 (2020), where multiple kind of emitters are combined through a multi-armed bandit approach.

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. (2020). Multi-emitter map-elites: Improving quality, diversity and convergence speed with heterogeneous sets of emitters. *arXiv preprint arXiv:2007.05352*.

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., Mollard, Y., and Oudeyer, P.-Y. (2017). Intrinsically motivated goal exploration processes with automatic curriculum learning. *arXiv preprint arXiv:1708.02190*.

Grillotti, L. and Cully, A. (2021). Unsupervised behaviour discovery with quality-diversity optimisation. *arXiv preprint arXiv:2106.05648*.

Hansen, N. (2016). The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*.

Hare, J. (2019). Dealing with sparse rewards in reinforcement learning. *arXiv preprint arXiv:1910.09281*.

Hinton, G. E. and Zemel, R. S. (1994). Autoencoders, minimum description length and helmholtz free energy. In *Advances in neural information processing systems*, pages 3–10.

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. *arXiv preprint arXiv:2011.02669*.

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.

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.

Pearson, K. (1895). Vii. note on regression and inheritance in the case of two parents. *proceedings of the royal society of London*, 58(347-352):240–242.

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.

Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Wiele, T., Mnih, V., Heess, N., and Springenberg, J. T. (2018). Learning by playing solving sparse reward tasks from scratch. In *International Conference on Machine Learning*, pages 4344–4353. PMLR.

Salehi, A., Coninx, A., and Doncieux, S. (2021). Br-ns: an archive-less approach to novelty search. *arXiv preprint arXiv:2104.03936*.

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.