



**HAL**  
open science

# Learning in Sparse Rewards settings through Quality-Diversity algorithms

Giuseppe Paolo

► **To cite this version:**

Giuseppe Paolo. Learning in Sparse Rewards settings through Quality-Diversity algorithms. Computer Science [cs]. Sorbonne Université, 2021. English. NNT : . tel-03707344v1

**HAL Id: tel-03707344**

**<https://hal.science/tel-03707344v1>**

Submitted on 25 Feb 2022 (v1), last revised 28 Jun 2022 (v2)

**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.

---

# Learning in Sparse Rewards settings through Quality-Diversity algorithms

**Thèse de Doctorat de Sorbonne Université**

Spécialité : Informatique (EDITE)

Présentée par : M. Giuseppe Paolo

Soutenue le : **05 Novembre 2021**

Pour obtenir le grade de  
Docteur de Sorbonne Université

*Rapporteurs*

Jean-Baptiste MOURET  
Salima HASSAS

*Examineurs*

Antoine CULLY  
Sylvain LAMPRIER  
Natalia DIAZ RODRIGUEZ

*Directeur*

Stéphane DONCIEUX

*Co-encadrants*

Alban LAFLAQUIÈRE  
Alexandre CONINX



# English Abstract

Embodied agents, both natural and artificial, can learn to interact with the environment they are in through a process of trial and error. This process can be formalized through the Reinforcement Learning framework, in which the agent performs an action in the environment and observes its outcome through an observation and a reward signal. It is the reward signal that tells the agent how good the performed action is with respect to the task. This means that the more often a reward is given, the easier it is to improve on the current solution. When this is not the case, and the reward is given sparingly, the agent finds itself in a situation of sparse rewards. This requires a big focus on exploration, that is on testing different things, in order to discover which action, or set of actions leads to the reward. RL agents usually struggle with this. Exploration is the focus of Quality-Diversity methods, a family of evolutionary algorithms that searches for a set of policies whose behaviors are as different as possible, while also improving on their performances. In this thesis, we approach the problem of sparse rewards with these algorithms, and in particular with Novelty Search. This is a method that, contrary to many other Quality-Diversity approaches, does not improve on the performances of the discovered rewards, but only on their diversity. Thanks to this it can quickly explore the whole space of possible policies behaviors.

The first part of the thesis focuses on autonomously learning a representation of the search space in which the algorithm evaluates the discovered policies. In this regard, we propose the *Task Agnostic eXploration of Outcome spaces through Novelty and Surprise (TAXONS)* algorithm. This method learns a low-dimensional representation of the search space in situations in which it is not easy to hand-design said representation. TAXONS has proven effective in three different environments but still requires information on when to capture the observation used to learn the search space. This limitation is addressed by performing a study on multiple ways to encode into the search space information about the whole trajectory of observations generated during a policy evaluation. Among the studied methods, we analyze in particular the mathematical transform called *signature* and its relevance to build trajectory-level representations.

The manuscript continues with the study of a complementary problem to the one addressed by TAXONS: how to focus on the most interesting parts of the search space. Novelty Search is limited by the fact that all information about any reward discovered during the exploration process is ignored. In our second contribution, we introduce the *Sparse Reward Exploration via Novelty Search and Emitters (SERENE)* algorithm. This method separates the exploration of the search space from the exploitation of the reward through a two-alternating-steps approach. The exploration is performed through Novelty Search, but whenever a reward is discovered, it is exploited by instances of reward-based methods - called emitters - that perform local optimization of the reward. Experiments on different environments show how SERENE can quickly obtain high rewarding solutions without hindering the exploration



performances of the method.

In our third and final contribution, we combine the two ideas presented with TAXONS and SERENE into a single approach: *SERENE augmented TAXONS (STAX)*. This algorithm can autonomously learn a low-dimensional representation of the search space while quickly optimizing any discovered reward through emitters. Experiments conducted on various environments show how the method can i) learn a representation allowing the discovery of all rewards and ii) quickly exploit those rewards thanks to the emitters.

Throughout this thesis, we introduce methods that, while dealing with sparse rewards situations, lower the amount of prior information needed at design time. These contributions are a promising step towards the development of methods that can autonomously explore and find high-performance policies in a variety of sparse rewards settings. This could increase the range of applicability of existing approaches leading to more autonomous embodied agents.

# French Abstract

Les agents incarnés, qu'ils soient naturels ou artificiels, peuvent apprendre à interagir avec l'environnement dans lequel ils se trouvent par un processus d'essais et d'erreurs. Ce processus peut être formalisé dans le cadre de l'apprentissage par renforcement, dans lequel l'agent effectue une action dans l'environnement et observe son résultat par le biais d'une observation et d'un signal de récompense. C'est le signal de récompense qui indique à l'agent la qualité de l'action effectuée par rapport à la tâche. Cela signifie que plus une récompense est donnée, plus il est facile d'améliorer la solution actuelle. Lorsque ce n'est pas le cas, et que la récompense est donnée avec parcimonie, l'agent se retrouve dans une situation de récompenses éparses. Cela nécessite de se concentrer sur l'exploration, c'est-à-dire de tester différentes choses, afin de découvrir quelle action ou quel ensemble d'actions mène à la récompense. Les agents RL ont généralement du mal à le faire. L'exploration est le point central des méthodes de Qualité-Diversité, une famille d'algorithmes évolutionnaires qui recherche un ensemble de politiques dont les comportements sont aussi différents que possible, tout en améliorant leurs performances. Dans cette thèse, nous abordons le problème des récompenses éparses avec ces algorithmes, et en particulier avec Novelty Search. Il s'agit d'une méthode qui, contrairement à de nombreuses autres approches Qualité-Diversité, n'améliore pas les performances des récompenses découvertes, mais uniquement leur diversité. Grâce à cela, elle peut explorer rapidement tout l'espace des comportements possibles des politiques.

La première partie de la thèse se concentre sur l'apprentissage autonome d'une représentation de l'espace de recherche dans lequel l'algorithme évalue les politiques découvertes. A cet égard, nous proposons l'algorithme *Task Agnostic eXploration of Outcome spaces through Novelty and Surprise (TAXONS)*. Cette méthode apprend une représentation à faible dimension de l'espace de recherche dans des situations où il n'est pas facile de concevoir manuellement cette représentation. TAXONS s'est avéré efficace dans trois environnements différents mais nécessite encore des informations sur le moment où il faut saisir l'observation utilisée pour apprendre l'espace de recherche. Cette limitation est abordée en réalisant une étude sur les multiples façons d'encoder dans l'espace de recherche des informations sur la trajectoire complète des observations générées pendant une évaluation de politique. Parmi les méthodes étudiées, nous analysons en particulier la transformation mathématique appelée *signature* et sa pertinence pour construire des représentations au niveau de la trajectoire.

Le manuscrit se poursuit par l'étude d'un problème complémentaire à celui abordé par TAXONS : comment se concentrer sur les parties les plus intéressantes de l'espace de recherche. Novelty Search est limitée par le fait que toute information sur une récompense découverte au cours du processus d'exploration est ignorée. Dans notre deuxième contribution, nous présentons l'algorithme *Sparse Reward Exploration via Novelty Search and Emitters (SERENE)*. Cette méthode sépare l'exploration de l'espace de recherche de

l'exploitation de la récompense par une approche en deux étapes alternées. L'exploration est effectuée par Novelty Search, mais lorsqu'une récompense est découverte, elle est exploitée par des instances de méthodes basées sur la récompense - appelées émetteurs - qui effectuent une optimisation locale de la récompense. Des expériences sur différents environnements montrent comment SERENE peut obtenir rapidement des solutions à forte récompense sans nuire aux performances d'exploration de la méthode.

Dans notre troisième et dernière contribution, nous combinons les deux idées présentées avec TAXONS et SERENE en une seule approche : *TAXONS augmentés par SERENE (STAX)*. Cet algorithme peut apprendre de manière autonome une représentation à faible dimension de l'espace de recherche tout en optimisant rapidement toute récompense découverte grâce à des émetteurs. Des expériences menées sur différents environnements montrent comment la méthode peut i) apprendre une représentation permettant la découverte de toutes les récompenses et ii) exploiter rapidement ces récompenses grâce aux émetteurs.

Tout au long de cette thèse, nous introduisons des méthodes qui, tout en traitant des situations de récompenses éparses, réduisent la quantité d'informations préalables nécessaires au moment de la conception. Ces contributions constituent une étape prometteuse vers le développement de méthodes capables d'explorer et de trouver de manière autonome des politiques performantes dans une variété de situations de récompenses éparses. Cela pourrait augmenter le champ d'application des approches existantes et conduire à des agents incarnés plus autonomes.



# Acknowledgement

Completing a Ph.D. thesis is a huge task and the whole process is a though but incredible journey. Doing all of this just by myself would have been impossible, even more considering the situation created by COVID in these past years. I want to thank here all the people that, with their help and support, made reaching the end of this journey possible.

First and foremost, I want to thank my supervisors. They believed in me, giving me the opportunity to freely develop my research ideas. If I am here as a young scientist, it is without any doubt thanks to them.

I am incredibly proud of having done this under the supervision of Stéphane Doncieux. The support he gave me, and the patience shown in the most stressful moments, meant a lot. His advice has always been incredibly helpful and allowed me to grow as a scientist while keeping me on track towards the final goal.

A great part of the supervision came also from Alban Laflaquière. He proved to be a great manager and allowed me to have all the freedom I needed within SBRE while keeping a close eye on my work. His suggestions have always been of the highest quality, and I really enjoyed our discussions on the most disparate topics.

A big thanks also to Alexandre Coninx, whose supervision and suggestions helped me to get out of the many “local minima” that a Ph.D. student finds in its path.

Both ISIR and SBRE provided me with the right working environment and the resources needed to perform my research. Being part of these two institutions allowed me to meet many great colleagues and friends.

In particular, I want to thank the members of the AMAC equipe, the AI Lab, the Proto Lab, and the Expressivity team. The lunches, discussions, and activities we did together were a great part of the experience of my Ph.D. and I will always cherish these memories.

An important part of the life in the lab was also the open-ended learning group, whose immensely interesting and stimulating discussion never ceased to inspire me. I hope the spirit of discovery and discussion of the group will continue to inspire future PhDs and members of the lab.

I want to thank my family. Even if far away, and even if they did not always understand what I was doing, they never ceased to support me in any possible way. I wouldn't be here if it were not for them.

Living abroad, in the fantastic city that is Paris, I also made another family: Ginevra, Sara, Maura, Michel, Laura, Carmine, Marwen, Hugo, Alessan-

dro, Helena and all the amazing friends that made this experience incredible. The list would be too long to mention everyone, but you're all in my heart, and the time and "adventures" we had together will always be very fond memories for me.

Also thanks to Lisa, Désirée and Gabriele for the support (and the wifi hotspot) in the last moments of this manuscript redaction.

Finally, I want to thank the members of the Jury of my Ph.D. defense for accepting being here. Their opinion and judgment of my work will surely be a good conclusion of these past 3 years and help me in defining my future career.

Sometimes I feel this went all too fast.

And as someone wiser than me once said: "Dovrò soltanto reimparare a camminare"

**Thanks for everything!**





---

# Contents

<b>English Abstract</b>	<b>iii</b>
<b>French Abstract</b>	<b>v</b>
<b>Contents</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Background and related work</b>	<b>10</b>
2.1 Reinforcement Learning . . . . .	10
2.1.1 Markov Decision Process . . . . .	11
2.1.2 Exploration-exploitation trade-off . . . . .	13
2.1.3 Sparse rewards . . . . .	14
2.2 Evolutionary Algorithms . . . . .	18
2.2.1 Multi-objective optimization . . . . .	20
2.2.2 Searching for Diversity . . . . .	22
<b>3 TAXONS</b>	<b>31</b>
3.1 Introduction . . . . .	31
3.2 Related work . . . . .	32
3.3 Methodology . . . . .	34
3.3.1 Policy selection . . . . .	36
3.3.2 Search and Training . . . . .	38
3.4 Experiments . . . . .	38
3.4.1 Experimental setup . . . . .	38
3.4.2 Evaluation . . . . .	40
3.4.3 Results . . . . .	42
3.5 Conclusion . . . . .	45
<b>4 Signatures</b>	<b>48</b>
4.1 Introduction . . . . .	48
4.2 The signature transform . . . . .	49
4.2.1 Signature of a discrete path . . . . .	53
4.3 Signed Behavior Descriptor . . . . .	53
4.4 Experiments . . . . .	56
4.5 Results . . . . .	57
4.5.1 Exploration . . . . .	57



4.5.2	Rewards . . . . .	59
4.6	Conclusion . . . . .	60
<b>5</b>	<b>SERENE</b>	<b>63</b>
5.1	Introduction . . . . .	63
5.2	Emitters . . . . .	64
5.3	Method . . . . .	65
5.3.1	Exploration phase . . . . .	67
5.3.2	Exploitation phase . . . . .	68
5.4	Experiments . . . . .	72
5.5	Results . . . . .	74
5.5.1	Budgeting . . . . .	74
5.5.2	Exploration . . . . .	76
5.5.3	Exploitation . . . . .	77
5.5.4	Final archive distribution . . . . .	77
5.6	Conclusion . . . . .	80
<b>6</b>	<b>STAX</b>	<b>82</b>
6.1	Introduction . . . . .	82
6.2	Method . . . . .	83
6.2.1	Policy Selection . . . . .	84
6.2.2	Training of the autoencoder . . . . .	84
6.2.3	Reward exploitation in a learned space . . . . .	86
6.3	Experiments . . . . .	88
6.4	Results . . . . .	92
6.4.1	Exploration . . . . .	92
6.4.2	Exploitation . . . . .	94
6.4.3	Final archives distribution . . . . .	94
6.4.4	Exploration ablation studies . . . . .	96
6.4.5	Autoencoder training regime . . . . .	100
6.4.6	Learned behavior space . . . . .	103
6.5	Conclusion . . . . .	106
<b>7</b>	<b>Discussion</b>	<b>108</b>
7.1	Learning the behavior space . . . . .	108
7.1.1	Distractors . . . . .	110
7.1.2	Disentangled representations . . . . .	111
7.2	Focusing on the interesting parts of the search space . . . . .	111
7.3	Noisy environments . . . . .	112
<b>8</b>	<b>Conclusion</b>	<b>115</b>
	<b>Bibliography</b>	<b>120</b>



---

# Acronyms

**AE** autoencoder.

**BS** Behaviour Space.

**CNN** Convolutional Neural Network.

**DMP** Dynamic Movement Primitive.

**DoF** degrees of freedom.

**EA** Evolutionary Algorithm.

**ES** Evolution Strategies.

**GAN** Generative Adversarial Network.

**GEP** Goal Exploration Processes.

**HER** Hindsight Experience Replay.

**IM** Intrinsic Motivation.

**IMGEP** Intrinsically Motivated Goal Exploration Processes.

**LSTM** Long Short-Term Memory.

**MAB** Multi-Armed Bandit.

**MDP** Markov Decision Process.

**ME** MAP-Elites.

**MOO** Multi-objective optimization.

**NN** neural network.

**NS** Novelty Search.

**NSLC** Novelty Search with Local Competition.

**QD** Quality-Diversity.

**RL** Reinforcement Learning.

**RND** Random Network Distillation.

**SERENE** Sparse Reward Exploration via Novelty and Emitters.

**STAX** SERENE augmented TAXONS.

**TAXONS** Task Agnostic eXploration of Outcome space through Novelty and Surprise.

**VAE** Variational Autoencoder.



---

# List of Figures

Figure 2.1	Reinforcement Learning cycle . . . . .	11
Figure 2.2	MDP example . . . . .	12
Figure 2.3	Sparse Reward environment . . . . .	15
Figure 2.4	Evolutionary algorithms cycle . . . . .	19
Figure 2.5	Pareto front . . . . .	21
Figure 2.6	NSGA-II policy ordering . . . . .	22
Figure 2.7	NSGA-II policy selection . . . . .	23
Figure 3.1	TAXONS . . . . .	34
Figure 3.2	Pixel-wise distance example . . . . .	36
Figure 3.3	TAXONS test environments . . . . .	39
Figure 3.4	TAXONS AE structure . . . . .	40
Figure 3.5	TAXONS final archives . . . . .	43
Figure 3.6	TAXONS coverage results . . . . .	44
Figure 4.1	Geometric representation of signature . . . . .	51
Figure 4.2	Signature examples . . . . .	54
Figure 4.3	CollectBall environment . . . . .	56
Figure 4.4	Signature coverage results . . . . .	58
Figure 4.5	Signature dimensionality experiments - coverage results . . . . .	59
Figure 4.6	Signature reward results . . . . .	60
Figure 5.1	SERENE . . . . .	66
Figure 5.2	SERENE sets . . . . .	67
Figure 5.3	SERENE test environments . . . . .	72
Figure 5.4	SERENE budgeting results . . . . .	75
Figure 5.5	SERENE coverage results . . . . .	76
Figure 5.6	SERENE reward results . . . . .	78
Figure 5.7	SERENE final archives . . . . .	79
Figure 6.1	Example of multiplication of reward areas . . . . .	88
Figure 6.2	Curling environment . . . . .	90
Figure 6.3	HardMaze environment . . . . .	90
Figure 6.4	Redundant Arm environment . . . . .	91
Figure 6.5	STAX AE structure . . . . .	92
Figure 6.6	STAX coverage results . . . . .	93
Figure 6.7	STAX reward results . . . . .	95

Figure 6.8	STAX final archives	97
Figure 6.9	STAX ablation experiments - coverage results	98
Figure 6.10	STAX ablation experiments - reward results	99
Figure 6.11	STAX learned BS experiments - coverage results	101
Figure 6.12	STAX learned BS experiments - final archives	102
Figure 6.13	STAX AE reconstruction	103
Figure 6.14	STAX AE behavior descriptors	105

---

# Introduction

An embodied agent is any agent situated in an environment with which it interacts. The natural world around us is full of this kind of agents: not only humans, but animals, plants, fungi, bacteria can all be considered embodied agents. They all act and interact with the world they are in, the environment, by following some kind of policy. This policy can be either innate, in which case it is usually referred as instinct [1, 2], or learned during the life of the agent itself. In both situations, the policy dictates which actions the agent should perform on the environment as a reaction to the state the environment or the agent itself are in. Up until very recently, the only existing type of embodied agents were biological beings, evolved through natural evolution in the enormous variety of living beings known today [3]. This is not the case anymore, thanks to the many research advancements that have lead to the development of artificial - albeit still rudimentary - embodied agents.

Humans have long dreamt of creating artificial agents capable of interacting with the world they are in. The first accounts of these ideas date back to ancient Greece: in Homer's Iliad are described the creation of artificial agents by both gods and humans [4]. Similar themes are present also in Chinese and Egyptian mythology [5]. The appearance of stories like this in cultures so distinct and far apart demonstrates how great for humans is the desire to create artificial agents. A desire that continued through the Middle Ages and the Renaissance, when many "embodied agents" were built and displayed by scientists and engineers all around the world [6]. These machines were known as *automata*, a word coming from ancient Greek meaning "acting on one's own will", a name highlighting how they could interact with the world by following a policy, their "will". Notwithstanding the promise given by the name, and the fascination these machines were provoking in people at the time, automata were nothing more than complex mechanism, much more similar to a clock than to an agent capable of reacting and adjusting to the state of the world. The policy governing them was designed to perform only a limited set of actions in a way that could give the illusion of will. Moreover, due to said policy being part of the hardware design, it could not be modified without rebuilding the whole machine.

Subsequent technological developments, an in particular the invention of the computer, allowed the creation of ever more sophisticated agents, capable of interacting with the world in multiple and better ways. Agents of this kind are known today as *robots*, a word coming from the Slavic languages expres-

sion for forced labor [7]. Contrary to automata, the word robot stresses the fact that these are machines, deprived of any will, that just follow a set of instructions, the policy. At the same time, robots have an important advantage on ancient automata in the fact that they have sensors. This enables the creation of agents capable of dealing with a much bigger range of situations. Thanks to this, the development of robots allowed the automation of many labor-intensive tasks. An example of this is the introduction of robots in assembly lines, warehouses and other environments requiring heavy and repetitive tasks, allowing the automation and the increase in production efficiency of these systems [8, 9, 10]. In the future, even more advantages for human societies are predicted to come thanks to robots [11]. At the same time, these are very controlled and well defined systems, for which relatively simple policies can be hand-designed by the engineers. This is not the case for more complex and difficult to control settings, for which the policy designer must take into account all possible interactions between the robot and the elements of the environment. A robot performing inspection of an industrial plant [12], has to deal with uneven terrain, doors to open, objects to move or navigate around. There is an almost infinite amount of different situations to deal with. For this kind of problems, learning the controller policy is an effective alternative approach to hand-designing it. This allows the agent to adapt its strategy to the task at hand, while moving the engineer's design effort on the training process. The advantage of this approach is its generalizability: the same training process can be used to learn policies for different tasks. There are multiple ways in which a policy can be learned, but they typically consist in algorithms optimizing a performance metric measuring how well the learned policy can accomplish the desired goal. Algorithms of this kind belong to the field of Artificial Intelligence, and as many of the methods in this discipline they take huge inspiration from natural processes and the way animals learn. Depending on which natural mechanics they are based on, policy learning methods can be grouped in two families: [Reinforcement Learning \(RL\)](#) and [Evolutionary Algorithms \(EAs\)](#).

## Reinforcement Learning

[RL](#) is a framework that can be used for learning policies able to solve a given task through a process of "trial-and-error" [13]. It is inspired by the way animals learn how to solve tasks: try something and according to the outcome, learn to repeat or to avoid the same action in similar situations [14]. The origin of [RL](#) can in fact be traced back to Thorndike's Law of Effect [15] and Pavlov's studies on conditioned reflexes [16]. Both scientists studied how behaviors that were followed by positive outcomes were more likely to become established and be repeated in similar situations. A famous experiment in this regard is Pavlov's dog study [16, 17]. The experiment consisted in placing a dog in a room, in which some food is delivered every time a bell is rang. After few of these interactions, the scientist observed that the dog's salivation would increase whenever the bell was rang, as if the animal was in presence of food,

even if no food was delivered anymore. This shows how the dog learned a behavior, the salivation, whenever a given conditioning, the sound of the bell, happened; even if no food was given anymore. RL algorithms imitate the same mechanism in order to learn a policy, rendering them extremely flexible on the range of problems they can solve [18, 19, 20, 21]. A fundamental component of any RL system is the reward function, used to drive the training process towards learning a good policy. It is through this function that the engineer "communicates" to the agent the task it needs to solve. This means that a great part of the design effort has to be directed towards the definition of that function. In general, RL methods require the reward to be dense. This means that the reward function should give a feedback on every action the agent performs. Unfortunately, this is not always the case. When the reward is given only after multiple actions have been performed, or if a specific situation is met, the agent has to deal with a *sparse reward system*. Sparse rewards mean that in many of the states of the system there is no clear signal of what is the best course of action. In these settings, standard RL algorithms struggle to learn good policies, leading to poor performances or to no solution at all. Nonetheless, given the ubiquity of sparse rewards systems, being able to deal with them is fundamental. Even more so when the agent is in the real world. There are many factors rendering the design of a dense reward function difficult for real world problems. The task could be not well defined, too broad or complex, or require too much information in order to calculate a reward after each action. An example of this is a search-and-rescue mission [22]. While the task is well defined - explore the area and look for people in need of help - the design of a dense reward function is not easy. Awarding the agent for every discovered person would be simple, but such a function is incredibly sparse. To have a denser reward, the position of the dispersed people would have to be known in advance, but this, other than requiring too much information, would remove the search part from the search-and-rescue mission. This is an extreme scenario, but it is a good example of how agents capable of dealing with sparse rewards could help in addressing many difficult and dangerous situations.

Recently, many scientists focused their research efforts on proposing algorithms capable of solving sparse reward problems. This ranges from reward shaping, in which the reward function is modified in a way that can lead the agent to solve the task, [23], to intrinsic motivation, where the agent generates the reward by itself by maximizing another metric [24, 25]. Other methods include the agent learning from past experiences to reach self assigned goals [26], or solving of auxiliary tasks that can help in learning how to reach the main goal [27].

In general, when the reward is sparse the agent has no clue where to start to solve the task. In these situations a good strategy is to focus on *exploration*, to discover all the possible things that can be done in the environment. This strategy also applies to the previous example of a search-and-rescue mission: by focusing on exploring the environment, the agent can discover the missing people and thus be able to maximize its reward. In this thesis, we approach



the problem of sparse rewards with this idea in mind, studying and proposing a set of algorithms capable of performing efficient exploration in this kind of settings. Contrary to the methods discussed until now, this is done through the other family of policy learning methods mentioned before: **EAs** [28]. The reason behind this is the greater versatility of **EAs** algorithms thanks to them being gradient-free. Not having to calculate any gradient allows for their applications in a much wider range of situations, without the requirement to have a differentiable policy parametrization. At the same time, this comes at a price: **EAs** are slower than gradient-based methods in their optimization process.

## Evolutionary Algorithms

As the name indicates, **EA** are directly inspired by the natural evolution process described by Darwin [3]. In his work, Darwin described how, given a population of living beings in an environment, the elements better adapted to the environment have higher chances of reproduction, while the ones less suited are more likely to die prematurely. In time, the natural selection of fitter elements will lead to the development of a population of agents highly adapted to live in the environment in which it finds itself [3, 29].

A similar selection process is also applied by **EAs** when performing the search for policies. These algorithms work with a population of policies that are used to generate new policies through two operators: *mutation* and *crossover*. The first, inspired by the generic mutation happening in nature, randomly mutates some of the parameters of a policy to generate a new one. The crossover is instead inspired by sexual reproduction: the parameters of two or more policies are mixed to generate a new set of policies. The newly generated policies according to these operators are then evaluated in the environment. Among them, only the best ones are selected to form the next generation population, according to a given performance metric, usually called *fitness function*. Notwithstanding the different name, **EAs'** fitness function has the same role the reward function has in **RL**. For this reason, and given that in the literature the problem of sparse rewards is mainly defined with respect to **RL**, throughout this thesis we will refer to the fitness of a policy as to its reward. Contrary to **RL** algorithms, expecting a reward for every action performed, the performance evaluation done by **EAs** on their policies happens only at the end of their execution, rendering them better suited for sparse rewards systems. Moreover, thanks to the fact that these algorithms do not make any assumption about the fitness landscape they are in, they have proven useful in many domains, from circuit design [30] to the evolution of other artificial intelligence algorithms [31]. Nonetheless, researchers have been wondering why standard **EAs** cannot generate the same amount of diversity generated by the natural evolution process. These algorithms are very prone to converge to a single local minima, with the whole population being the same, a phenomenon known as *population collapse* [32, 33]. One of the possible culprits for this issue has been identified in the fitness function [34].

If not properly designed, this function can lead the search astray or towards dead ends. Moreover, relying on the fitness function to drive the search can be problematic in situations in which this function is extremely sparse. In these scenarios it can happen that no policy in a whole population can get any reward at the end of its evaluation, rendering the search for a solution difficult. To address these problems, Lehman and Stanley proposed a novel take on the design of **EAs** by introducing the **Novelty Search (NS)** algorithm [34, 35]. This algorithm works by completely ignoring the reward and just focusing on exploration, looking for *novel behaviors*. The idea behind the **NS** approach is that while the amount of behavior of a certain complexity is limited, many policies in the search space can express the same behavior. By only focusing on novel behaviors the search can thus discover more complex and diverse ways of acting, possibly finding a solution to the task. This allows the algorithm to return a whole collection of policies, each one with a significantly different behavior from the others. According to how the policies and their behaviors are defined, this collection can then be used in many different ways [36, 37, 38, 39]. The authors have shown that this way of performing the search for policies, rather than optimizing a reward function, allows the discovery of solutions for problems in which many other reward-based algorithms get stuck [35].

### Quality-Diversity algorithms

The introduction of **NS** extended the range of problems to which **EAs** can be applied, sparking a renewed interest in using methods from this field for learning policies. This led to the development of many similar algorithms that, rather than looking for the best solution to a problem, can perform *divergent search* and find a set of many different solutions [40, 41, 42, 43, 44, 45]. Some of these algorithms are designed to not only optimize the diversity of the discovered set of policies, but also their quality towards the given goal. Due to this characteristic these methods are usually referred to as **Quality-Diversity (QD)** algorithms [41, 42]. By discovering a whole set of policies rather than a single one, these algorithms make the embodied agent more adaptable to different situations and tasks. This has been shown by using **MAP-Elites (ME)** [43], a well known **QD** method, to train a hexapod to walk and to quickly adapt to damages to its legs, even if no damage was present at training time [46].

The generation of multiple policies, and the great exploration ability divergent search algorithms have made us choose them as an approach to perform policy search for sparse rewards. An overview of the literature on these methods and a detailed description of how both **RL** and **EAs** work will be given in Chapter 2.

Notwithstanding their advantages, divergent search algorithms are still limited in many ways. The most notable limitation is the way the diversity of the policies' behaviors is calculated. This is done in a space, called the **Behaviour Space (BS)**, in which the behaviors are represented and that is

usually hand-designed by the engineer setting up the learning system. While this can help tailor the solution to the problem at hand, it requires a significant amount of prior knowledge about the features of the system, the robot and the task itself. The search will also be constrained by the biases of the designer's choices or by the need to have access at runtime to informations difficult to extract. For example, in the case of a robot learning to throw a ball in different ways [47], the ball position needs to be properly tracked in order to estimate where it touches the ground. Not doing so would make it difficult to distinguish between different behaviors. This can strongly limit the range of application of QD algorithms.

## Learning the behavior space

In light of the problems just discussed, Chapters 3 and 4 explicitly address the following question:

*How to remove the requirement of hand designing the BS for NS, and QD algorithms in general?*

Having an algorithm capable of autonomously learning the BS in which the search is performed could greatly help in this direction. A way to address the issue is proposed in Chapter 3, with the introduction of the **Task Agnostic eXploration of Outcome space through Novelty and Surprise (TAXONS)** algorithm [48]. This is a divergent search algorithm based on NS and designed to build in parallel, and in an unsupervised way, both a collection of diverse policies and the space in which the behavior of said policies is represented. It does so by encoding the high-dimensional observation of the final outcome generated by a policy evaluation into a lower dimensional representation through an **autoencoder (AE)** [49]. The conducted experiments show that **Task Agnostic eXploration of Outcome space through Novelty and Surprise (TAXONS)** can efficiently explore the whole space of possible behaviors in the tested environments.

The evaluation of a policy behavior in **TAXONS** is done with respect to its final outcome. To reach this outcome, the system traverses a whole trajectory of states during the evaluation of a policy, that are ignored by the algorithm. However, in some situations, considering only the final outcome is not enough to properly explore. Let us consider a robot learning how to throw an object against a wall. A good way to distinguish between the behaviors of different policies is by measuring the different positions where the object hits the wall. At the same time, without knowing the exact moment in which this happens, it is difficult to determine which state - from the trajectory of traversed states - to use to perform this measurement. In Chapter 4, a possible way of dealing with this problem is presented: the signature transform. This is a mathematical object consisting of an infinite series of integrals encoding a stream of data into a vector representing the geometrical properties of this stream [50, 51,

52]. Thanks to its many positive properties, the signature transform has been used in the field of machine learning to encode sequences of data in a fast and easy way [53, 54]. Given that the signature can encode a sequence of data into a single vector, it is a good candidate to encode the whole trajectory of states traversed by the system during the evaluation of a policy. This method is compared against simpler approaches that can help in removing the limitation of working only with the final outcome. The results show that these simpler methods are as effective as the signature in this regard. For this reason, we choose to use one of those simpler approaches in Chapter 6.

## Exploiting sparse rewards

After addressing the issue of reducing the amount of prior information about the search space and removing the limitation of hand-designing the BS, the manuscript continues by focusing on another important question:

*How to take advantage of the rewards discovered during the search performed by NS?*

NS is limited by the fact that all information about any reward discovered during the exploration process is discarded. Even if sparse, the reward is extremely useful in steering the search towards more interesting areas of the search space. To address this problem, the **SparsE Reward Exploration via Novelty and Emitters (SERENE)** algorithm is introduced in Chapter 5. This method combines the exploration abilities of NS with the exploitation of the reward provided by *emitters*, instances of reward based **EAs** performing local exploration [55, 56]. By combining these methods, SERENE can separate the exploration from the reward exploitation in an alternating two-steps process. The algorithm can then seamlessly adapt to a wide range of situations, from settings in which the reward is present in multiple areas of the search space to ones in which it is not present at all. Similarly to other divergent search algorithms, SERENE returns a collection of policies. This collection is divided in two groups: one containing the high performing policies optimized by the emitters and one containing the set of diverse policies usually returned by NS. The conducted experiments show how this approach can quickly explore the search space and optimize policies in settings of sparse rewards; even when multiple reward areas are present in the search space.

The **TAXONS** and **SERENE** methods proposed in Chapters 3 and 5 address complementary problems of NS: the hand-design of the BS and the exploitation of possible rewards found during the search. The final contribution of this manuscript is presented in Chapter 6: the **SERENE augmented TAXONS (STAX)** algorithm. This is a method merging ideas from previous chapters to address both problems at the same time. It does so by taking advantage of the representation learning abilities of **TAXONS** to drive the search in the two-steps process of **SERENE**. At the same time, the limitation present in **TAXONS** due to only observing the outcome of a policy to

describe its behavior is removed thanks to the lessons learned in Chapter 4. Experiments show that **STAX** can properly learn a good representation of the behavior space, discovering and quickly exploiting all the rewards present in the environment. This allows to have an algorithm capable of dealing with sparse reward environments, with minimal prior information about the task and the environment in which the embodied agent operates.

**STAX** represents the culmination of the work conducted in this thesis, that started with the identification of the sparse rewards problem and the possible ways of dealing with it. The advantages and the shortcomings of the approaches introduced throughout this manuscript are discussed in Chapter 7, highlighting the new and exciting possible research directions arising. The manuscript concludes with Chapter 8, providing a final overview of the work developed during this thesis.

As a recap, the work in this manuscript focuses on *how to deal with sparse rewards settings*. A good strategy to use in these situations is to focus on *exploration*, which **QD** algorithms, and **NS** in particular, are explicitly designed to do. At the same time, these methods perform the search in an *hand-defined* space, which can be a strong limitation in certain situations. The first presented contributions address this issue by *autonomously building a behavior space*, in order to reduce as much as possible the amount of prior information given at design time. The manuscript continues by highlighting how **NS** discards all informations about the most interesting parts of the search space by ignoring all rewards. The second presented contribution focuses on this limitation by introducing a method that augments **NS** with the ability to *exploit any reward discovered in the search space*. This is done without hindering the exploration performed by **NS**. Finally, these two contributions are merged by introducing a method that can *autonomously build a behavior space* while also *focusing on any discovered reward without hindering exploration*.



---

# Background and related work

---

## Chapter content

<b>2.1 Reinforcement Learning</b>	<b>10</b>
2.1.1 Markov Decision Process	11
2.1.2 Exploration-exploitation trade-off	13
2.1.3 Sparse rewards	14
<b>2.2 Evolutionary Algorithms</b>	<b>18</b>
2.2.1 Multi-objective optimization	20
2.2.2 Searching for Diversity	22

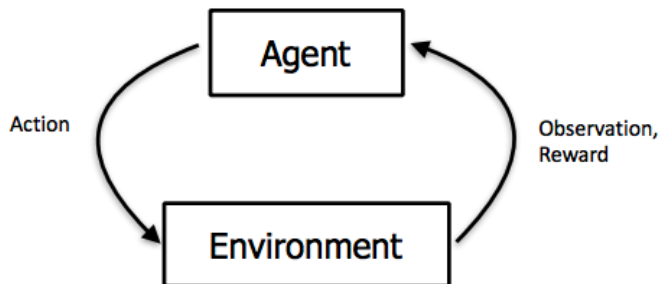
---

This chapter introduces the different research topics and algorithms on which this thesis builds. Along with the overview of the theory, there will be a discussion of the related approaches and the current state of the art. The chapter starts with an overview of [Reinforcement Learning \(RL\)](#) and the framework in which it operates when performing policy search for embodied agents. This will help in properly framing the issue of sparse rewards and describing some of the methods used to approach the problem. From there the chapter will describe how an [Evolutionary Algorithm \(EA\)](#) works and how, thanks to [Novelty Search \(NS\)](#), it is possible to overcome some of the limitations of classical [EAs](#) when optimizing policies. As said in [Chapter 1](#), [NS](#) sparked the development of a new family of [EAs](#) focusing on generating a set of diverse solutions, rather than a single optimal one. An overview of these methods will be given, with a detailed description of the most widely used algorithms in the domain.

## 2.1 Reinforcement Learning

[Chapter 1](#) discussed how an embodied agent can act in the environment it is in by following a policy. The policy is what governs the agent and defines how it acts in different situations. The goal of many learning algorithms for embodied agents is to learn a policy allowing the agent to solve a given task [[57](#), [58](#), [59](#)]. [Reinforcement Learning \(RL\)](#) [[13](#)] is a branch of machine learning that can be used for this. The focus of the [RL](#) framework is in fact to learn what actions an agent has to perform in an environment in order to maximize

a reward function. The learning is done through a trial-and-error approach in which, at each time step, the agent performs an action in the environment and observes the outcome of said action, expressed through an observation and a reward. The agent will then use this observation and reward to select the next action. The process is represented in Fig. 2.1.



**Figure 2.1:** *Reinforcement Learning cycle*

RL has proven to be a powerful and generic framework to model and address problems in many different fields, from resource management [18] and cross-light control [19], to robotics [20] and even chemistry [21]. In order to properly address a problem through an RL algorithm, the problem needs to be formulated as a **Markov Decision Process (MDP)**.

### 2.1.1 Markov Decision Process

A **Markov Decision Process (MDP)** is a time-discrete stochastic control model that can be used to describe a decision process with possibly random outcomes [60]. Thanks to its generality, it can be used to model problems in many fields, from robotics to economy. For this reason it has been widely studied and multiple approaches capable of solving an MDP have been proposed [61, 13, 62].

The main components of an MDP that need to be specified in order to formulate a problem in this framework are the following:

- the set of states  $S$ . It represents all the possible states  $s \in S$  in which the system can be in. The set can be either discrete or continuous;
- the set of actions  $A$ . It contains all the actions  $a \in A$  that the agent can perform on the environment. As with  $S$ ,  $A$  can also be either discrete or continuous;
- the state transition probability function  $P_a(s', s)$ . This function determines the probability of the system transitioning from state  $s$  at time  $t$  to state  $s'$  at time  $t + 1$  due to the agent performing action  $a$ :  $P_a(s', s) = Pr(s_{t+1} = s' | a_t = a, s_t = s)$ ;



- the reward function  $R_a(s', s)$ . This function determines the immediate reward the agent receives when transitioning in state  $s'$  from state  $s$  due to performing action  $a$ .

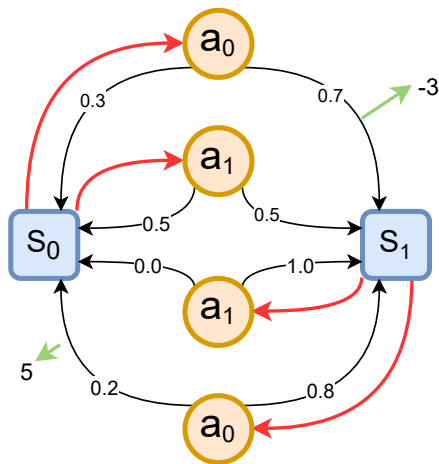
An important aspect of a problem formulated through an **MDP** is the fact that the state transition function has to respect the *Markov property*. It states that given a state  $s_t$  and an action  $a_t$  the probability that the system moves to state  $s_{t+1}$  is independent from all previous states and actions. Respecting the Markov property allows to greatly simplify the problem and its solution, thus many algorithms dealing with **MDPs** assume that the system respects it.

Finally, the *policy function* can be defined as  $\pi(s_t) = a_t$ . This function determines the action  $a_t$  to perform at time  $t$  when the system is in state  $s_t$ . The goal of a learning system acting on an **MDP** is to find the best policy  $\pi(\cdot)$  such that it maximizes the expected discounted sum over a potentially infinite horizon:

$$\mathbb{E} \left[ \sum_{t_0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) \right], \quad (2.1)$$

where  $\gamma^t \in [0, 1]$  is a discount factor for future rewards. The closer  $\gamma$  is to 1, the more importance is assigned to the future, the closer it is to 0, the more shortsighted the agent will be with respect to the reward.

Fig. 2.2 shows a simple **MDP**. In this example, the system can find itself in two possible states, shown in blue:  $[s_0, s_1]$ . At the same time, the agent can perform two actions, depicted in orange:  $[a_0, a_1]$ . The selection of an action while being in a state is represented by the red arrows. Each action makes the system transition to either one of the two states according to the probability indicated over each black arrow. In this system, the agent can obtain two rewards, indicated by the green arrows. Let us consider the case in which the system is in state  $s_1$ . The agent can perform either action  $a_0$  or action  $a_1$ . By performing action  $a_0$ , the system can go either back to state  $s_1$ , with probability 0.8, or go to state  $s_0$ , with probability 0.2. In this last case, the agent will receive a reward of 5.



**Figure 2.2:** Simple **MDP**. The two states in which the system can be are represented in blue, while the two possible actions are in orange. In each state the agent can select either one of the two actions by following the red arrow. This will cause the system to transition to another state according to the probabilities indicated over each black arrow. The two possible rewards are indicated by the green arrows.

Finding optimal policies for **MDPs** is not easy, even for the simple example shown in Fig. 2.2. For this reason, a lot of research has addressed these

problems, leading to the introduction of multiple methods capable of solving them [13, 59, 63, 64]. These methods work by calculating the value of a state  $s$  through the value function  $V(s)$  and updating the policy  $\pi(s)$  with respect to  $V(s)$  [13]. The value function is the expected return when starting from state  $s$  and following policy  $\pi$  and describes how good it is to be in a given state:

$$V(s) = \mathbb{E}_{\pi, P_a} [R(s'|s) + \gamma V(s')]. \quad (2.2)$$

The policy is updated with respect to the value of a state by selecting the action leading to the highest valued next state:

$$\pi(s) = \operatorname{argmax}_a \left\{ \sum_{s'} P(s'|s, a) (R(s'|s, a) + \gamma V(s')) \right\}. \quad (2.3)$$

As it can be seen from Fig.2.2, how good an action is strongly depends from the state the system is in. The reward obtained by performing action  $a_0$  in state  $s_0$  is different than the one obtained if the system was in  $s_1$ . This important aspect is evaluated through the Q-value function  $Q(a, s)$ . This function represents the *state-action value*, that is the value of performing an action  $a$  in a state  $s$  and is defined as:

$$Q_{\pi, P_a}(s, a) = \mathbb{E} [R(s'|s) + \gamma \mathbb{E}_{a \sim \pi} Q(s', a|s)]. \quad (2.4)$$

Different algorithms use these functions in different ways, but the final goal remains the same: discover the policy leading to the highest reward.

### 2.1.2 Exploration-exploitation trade-off

Always selecting the action according to Eq. (2.3), can easily lead the algorithm to get stuck in local minima. This would prevent it to find an optimal solution to the problem. The reason behind this is that, by always choosing the action that leads to the highest reward, the agent is only *exploiting* the knowledge it already has, not collecting new informations about the environment. This strategy is called a *greedy* strategy and it can prevent the discovery of other states and action combinations that can be more rewarding.

To discover new situations it is important to *explore* by testing different actions and visiting different states. At the same time, it is not given that higher rewarding situations can be found, so focusing too much on exploration rather than exploitation can be a waste of time and resources. It is then important to find a good balance between exploration and exploitation. This problem is usually referred as *the exploration-exploitation trade-off* [13]. There are multiple strategies that have been proposed to deal with it [65, 13] due to the fact that it is a fundamental problem in many learning settings. Among them, a very simple but well known one is the  $\epsilon$ -*greedy* algorithm [13].

The idea behind this method is simple: each time the agent has to choose an action, it selects the best action with probability  $1 - \epsilon$  or a random one

among the possible actions with probability  $\epsilon$ . The balance between the exploration and exploitation can be easily decided by changing the value of  $\epsilon$ . To only focus on exploration is enough to set  $\epsilon = 1$ , while a value of  $\epsilon = 0$  makes the agent completely greedy. By carefully setting  $\epsilon$ , this strategy allows to easily exploit what the agent already knows, but also to explore the environment and gather new information.

Given what has been discussed until now, it is possible to notice how the reward function  $R(\cdot)$  plays a fundamental role in the way RL algorithms operate. It is this function that is used to calculate both the value of each state  $s$  and to select the action to perform at each given step. This means that the reward function can be used by the designer of the problem to communicate to the algorithm which goal it needs to achieve. At the same time, to learn a good policy capable of obtaining the maximum reward possible, most RL methods expect a dense reward function. This implies that  $R(\cdot)$  needs to provide a relevant feedback on every single action the agent can perform on the environment. If the reward function rarely provide its feedback, it is defined as being a *sparse reward* [13, 66].

### 2.1.3 Sparse rewards

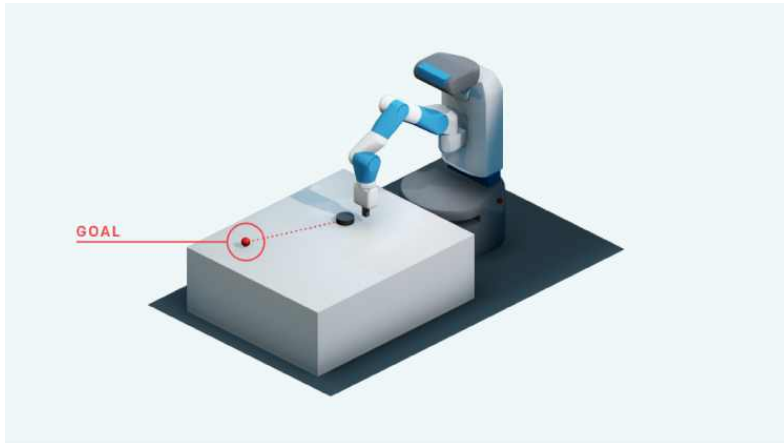
Given the importance of the role played by the reward function in learning a good policy, situations of sparse rewards can be extremely difficult to deal with [66]. The *ideal situation* in which to apply a RL algorithm is one in which the reward is well defined for each state and proportional to how beneficial that state is to reach the goal. Unfortunately, for many problems and in many real world settings this is not the case. Often the reward tends to be extremely sparse. In these situations, it can happen that the agent never experiences any reward at all, making learning a good policy impossible.

An example of a sparse reward system is given in Fig. 2.3, where a robotic arm has to push the black puck towards the goal, in red. The reward is given only if the puck reaches the target. This makes it very sparse: only one state among all the possible ones the system can be in would generate a reward. Another possible approach is to give the reward proportionally to how close the puck is to the goal. At the same time, this would require a precise measurement of its position at each time-step, requiring a more complex setup. Moreover, there can be other situations in which simple workarounds like this are not possible; the search-and-rescue example given in Chapter 1 is one of those. To have embodied agents capable of efficiently dealing with real world problems, approaches that can overcome sparse rewards problems needs to be developed.

For these reasons, many researchers have focused on sparse rewards situations, leading to the development of many different approaches.

### Reward shaping

The most simple way of dealing with a sparse rewards situation is by performing *reward shaping* [23]. This consists in augmenting the original reward



**Figure 2.3:** *Sparse reward environment from OpenAI Gym [67].*

function with additional features that can help the agent solve the task. Giving the reward depending on the distance between the puck and the goal, as discussed for the robot arm example, is a form of reward shaping. This approach has proven useful in many situations [68, 69, 70, 71, 72]. Notable is the work conducted by OpenAI on the Dota2 [73] game, in which the authors managed to train an agent to reach superhuman performances [70]. Nonetheless, reward shaping comes with multiple shortcomings. The new features added to the reward require huge amount of prior knowledge about the system and the task. Moreover, if not properly designed, this reward can introduce bias into the problem, leading the agent astray and preventing it to efficiently solve the problem.

### Self-assigning goals

Another approach is to have the agent self assign goals in order to train itself. This can be done by taking advantage of previously encountered situations, as done with [Hindsight Experience Replay \(HER\)](#) [26]. The main idea of the method is to store the state transitions in a *replay buffer*, even if no reward has been achieved. These stored transitions then are used by the algorithm to learn how to reach a goal, even if the given goal is not the one needed to solve the task. A similar goal relabeling approach is also used in the works from Levy et al. [74] and Nair et al. [75]. The first method takes advantage of hierarchical [RL](#), an approach that learns a hierarchy of policies in which at each step high-level policies select low-level policies to perform the task [74]. As for [HER](#), the self-assigned goals are sampled from the collection of already visited states. On the contrary, the second approach takes advantage of an unsupervised representation learning algorithm to generate the targets to reach [75]. In this method the visited states are not collected into a buffer, but are used to learn a compressed representation through a [Variational Autoencoder \(VAE\)](#) [76] from which the goals are then sampled. This approach increases the exploration abilities of the algorithm, allowing it to reach states not yet

visited by the agent.

A similar strategy to deal with sparse rewards is the one used by Florensa and his colleagues [77], sampling the goals from a **Generative Adversarial Network (GAN)** [78] rather than from the latent space of a **VAE**. The adversarial approach allows the agent to assign itself goals that become more complex with time. This strategy creates a sort of curriculum that continuously pushes the boundaries of the agent’s abilities. The strategy of using a curriculum to progressively increase the difficulty of the tasks to solve is also used by Riedmiller and his colleagues [79]. In their work, the agent tries to solve auxiliary tasks that start very simple and become more complex with time. This is repeated until the agent can solve the main task. The capacity to solve auxiliary tasks grants the agent a lot of flexibility in the range of goals that can be achieved. However, in order for the curriculum of tasks to be meaningful towards reaching the desired goal, the simpler tasks need to be selected by the engineer before training the agent.

### **Intrinsic Motivation**

A completely different idea for approaching sparse rewards problems is **Intrinsic Motivation (IM)** [80, 81]. Directly inspired from developmental psychology, this approach consists in the agent generating its own learning signal, without expecting any reward from the environment. This is similar to how kids learn something driven by their own curiosity rather than expecting some external reward [82]. An action can be defined intrinsically motivated if it is performed with the goal of collecting more information about the outcome of the action itself, rather than with the expectation of obtaining a reward [80, 81]. Given this definition, there are multiple ways to provide **IM** to an agent [83, 80].

**Novelty** The agent can be rewarded if it reaches more novel states, where the novelty is obtained by calculating an estimation of how often a state has been visited. With discrete state spaces, this estimation can be calculated by simply counting the number of times the agent visited a given state [84, 85]. For state spaces that are too big or continuous, this approach is not possible, requiring different strategies. One of those strategies is the use of a density estimation model over the state space to generate what the authors call *pseudo-count* [86]. For instance, the **Random Network Distillation (RND)** method uses a couple of **neural networks (NNs)** to estimate the novelty of each visited state [87]. One of the two **NNs** remains untrained with random weights, while the other is trained to reproduce the output of the untrained one. The novelty of a state is then assumed to be proportional to the difference between the output of the two models. The idea behind this is that the more often a state has been visited, the more the trained network has been trained to reproduce the same output of the random **NN** with respect to that state. This leads to a lower difference between the outputs of the two models. On the contrary, for a state that has been less visited, the trained model will return an output that is less similar to the one of the non trained **NN**. This leads to

the identification of a more novel state.

**Empowerment** Another possible way of providing IM to an embodied agent is through *empowerment* [88, 89, 90]. Based on Information Theory [91], this approach pushes the agent to maximize its control over the environment. This can be done by maximizing the entropy of future states the system can move into while also minimizing the entropy of the future states the system can reach given the action [89]. The rationale behind this being that the more an agent can influence the system to move to different states, the more control it has on the environment. There have been other approaches to empowerment, but the method is still hindered by the complexity of calculating the empowerment metric [81]. Simplifying this calculation would require a model of the environment itself, as done by Mohamed and Rezende [92], but in many situations this is not feasible.

**Curiosity** In a different direction goes the notion of *curiosity* [93, 94, 95]. This consists in providing the agent with a transition model on what can happen in the environment. The error between the predictions of the agent and what actually happens in the environment is then used as a reward signal by the agent. This will push the agent to explore more and look for situations in which it does not know what will happen, hopefully discovering a solution for the task. Curiosity as an IM approach has been systematically studied in [25], showing an alignment between the curiosity objective and the hand-designed rewards present in many game environments. This leads to good performances for curiosity-based methods in the tested situations. The agent’s error can also be used when predicting the consequences of its own actions to drive the learning process [24]. The error is calculated in a feature space learned through a self-supervised inverse dynamics module. This approach offers the advantage of not having to deal with the many unimportant informations present in pixel space that can keep the error high, e.g. leaves moving in the background or small changes in luminosity. The idea of curiosity can also be mixed with the one of the agent self-assigning goals as done with **Intrinsically Motivated Goal Exploration Processes (IMGEP)** [96, 97, 98]. IMGEPs work by having the agent sample its own desired goals from a given goal space according to a given strategy. Among the possible sampling strategies, a powerful one is to have a **Multi-Armed Bandit (MAB)** [63] with the goal to maximize the competence of the agent in reaching the sampled goals. This approach allows the creation of a curriculum, starting from simpler targets and moving toward more complex ones. Contrary to other goal self-assigning methods, **IMGEPs** are usually based on a population approach, in which multiple policies are tested at the same time. This allows more flexibility and an easier recovery of discovered abilities compared to other approaches that do not use a population [98].

Other than through curiosity, **Goal Exploration Processes (GEP)** help in addressing sparse rewards settings thanks to the separation between the exploration of the search space and the exploitation of any possible discovered

reward. Forestier et al. [97] use such a method by firstly learning a goal-parametrized policy capable of reaching any goal from any state and then using this policy to solve the task. This method has also been extended in IMGEP-UGL [99], in which the searched goal space is learned in an unsupervised fashion from the environment observations thanks to a dimensionality reduction algorithm. Another method, proposed by Colas and his colleagues [100], performs a task agnostic exploration phase before learning an inverse policy on the task to solve. Similarly, GoExplore [101] deals with the problem by using a two-phases strategy. It starts by exploring as much as possible, without caring about the reward. This leads to the building of a set of interesting states and trajectories leading to these states. Then in the second phase, it chooses one of the trajectories to turn it into a robust policy.

As stated in Sec. 2.1.1, RL methods work best in situations of dense rewards, rendering the problem of sparse rewards complex to address. Another way to deal with the issue is to rely on a different family of policy search algorithms, requiring the reward signal to be provided only at the end of the policy evaluation: [Evolutionary Algorithms \(EAs\)](#).

## 2.2 Evolutionary Algorithms

A different approach in learning policies for embodied agents consists in using a [Evolutionary Algorithm \(EA\)](#) [102]. [EAs](#) are a family of optimization algorithms inspired by the theory of natural evolution described by Darwin in its work "On the origin of species" [3]. These algorithms take advantage of the concept of *survival of the fittest* to perform *direct policy search*. They work with a population of individuals - each one of the corresponding to a parametrized policy - that is randomly initialized at the beginning of the search. This population is then evaluated in the environment and the performance of the individuals in it is measured through a given *fitness function*. Notwithstanding the different name, this function has a similar role than one of the reward function in RL methods. The main difference between the two functions is that the fitness function returns the reward for the whole evaluation episode, rather than evaluating the time-steps. Similarly to what happens in natural settings, the individuals whose fitness is too low will be discarded. At the same time, the agents deemed fit to survive are selected and allowed to reproduce through some variation operators, generating a new population of individuals to evaluate. Each iteration of this process is called a *generation*.

The selection of the individuals that are used for the generation of the new population can be performed in multiple ways [102, 103]. The most common selection strategies are:

- **Roulette wheel selection:** the probability of selecting an individual is proportional to its fitness; the better the fitness, the higher chance for that individual to be chosen;
- **Tournament selection:** multiple tournaments are performed between individuals sampled from the population. The winner of each tourna-



ment is selected for reproduction. This method can be improved by performing, at the end of each tournament, a probabilistic selection of the individual to reproduce: select the first with probability  $p$ , the second with probability  $p * (1 - p)$ , the third with probability  $p * (1 - p)^2$ , etc. [104];

- **Elitist selection:** the new population is composed not only by the new individual generated through reproduction, but also by the best elements from the previous generation. This allows to preserve particularly good set of parameters.

By continuously selecting the fittest agents the algorithm can generate agents that reach higher and higher fitness scores, thus discovering better ways to solve the task. The evolution cycle, represented in Fig. 2.4, is repeated until a given termination condition is reached.

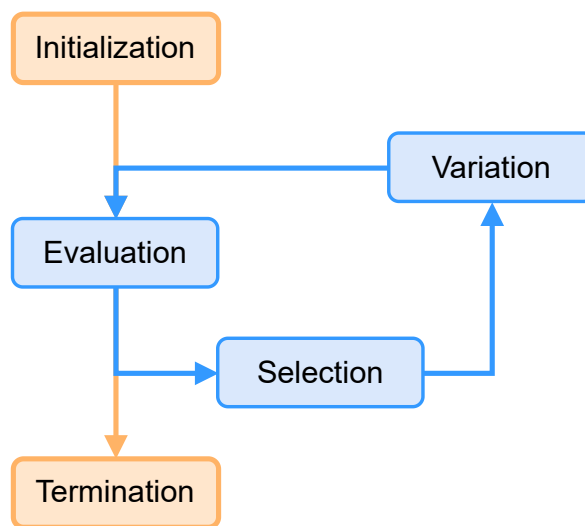


Figure 2.4: Evolutionary algorithms cycle [103]

## Genotype and Phenotype

Before defining how the creation of new individuals from the selected ones in the current population is performed, it is useful to define two important concepts for EAs: the *genotype* and the *phenotype*.

The *genotype* corresponds to any set of parameters used to define an individual. In the case of natural living beings the genotype is the DNA, encoding information about any aspect of the being. On the contrary, the *phenotype* is the expression of the information contained in the genotype. Keeping the parallel with the natural world, the phenotype expressed by the DNA can be the shape of the body, the color of the eyes or of the hair, or even the behavior of a living being. The mapping between the genotype and the phenotype of an agent strongly depends on the kind of setting and environment the algorithm is being applied to [103].



## Variation operators

In general, **EAs** act on the genotype in order to obtain and observe different phenotypes. This is done through two operators, used during the variation step of Fig. 2.4, to generate a new set of individuals from the existing one:

- **Mutation**: this operator is inspired by the genetic mutation happening in nature. It works by randomly changing some of the values of the set of parameters composing the genotype;
- **Crossover**: inspired by sexual reproduction, it combines the genotypes of two or more individuals to generate a new one.

These two operators are blind to the reward, meaning that the operations they perform do not depend on the reward. Nonetheless, they help the algorithm to continuously generate new individuals, properly exploring the genotype space in the search for solutions. At the same time, given that the crossover works with multiple individuals at once, it is not always straightforward to apply, depending on the structure of the genome and the way it is expressed in the phenotype. For this reason, many works in recent years tend to only use the mutation operator when generating new individuals [105].

The mutation and crossover operators are fundamentally stochastic in the selection of both individual and parameters on which to operate. While this renders the search less efficient compared to methods like **RL**, it also removes the need for the calculation of a gradient. Moreover, contrary to **RL** methods requiring the problem to be structured as an **MDP**, **EAs** can work with optimization problems structured as black-box functions [103]. The algorithm does not require any information about the internal structure of the problem or of the function it is optimizing, it just needs to provide an individual as possible solution and observe its final performance. This allows **EAs** to be applied to a wide variety of problems, from the design of buildings [106] to the generation of music [107] and images [108] and the creation of virtual creatures [40, 109]. **EAs** have also been used for the evolution of the topologies of **NNs** through neuroevolution [110, 111], or for the generation of policies used to control robots and embodied agents in general [112, 113, 46]. The only requirement in this regard is that the policy  $\pi(\cdot)$  has to be parametrized by a set of parameters  $\theta \in \Theta$ . These parameters correspond to the genotype of the controller policy, while the way the embodied agent acts in its environment is its phenotype.

### 2.2.1 Multi-objective optimization

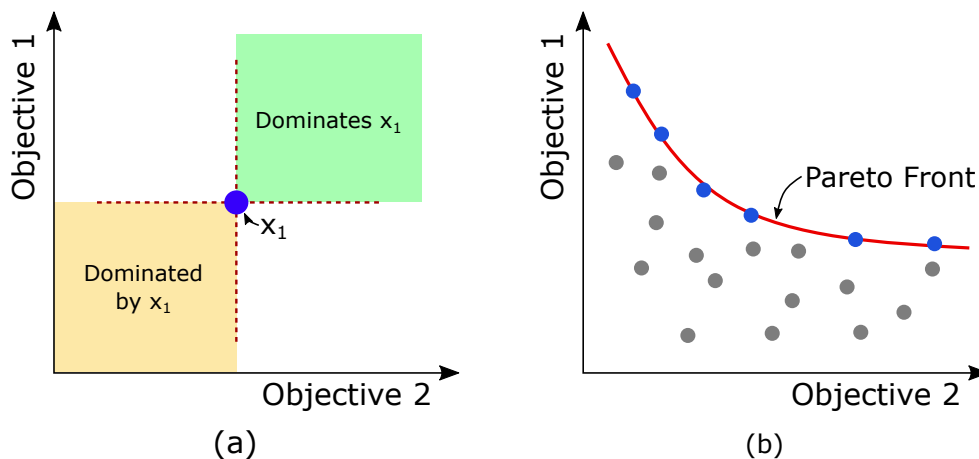
**EAs** can be applied also to **Multi-objective optimization (MOO)** problems, in which the method optimizes multiple objective functions at the same time [114]. An example of this can be seen in automated factories, in which the engineers have to optimize both the speed and the accuracy of the robots working on the assembly line. The solution of these kind of problems requires to find a good *trade-off* between the different objectives to optimize. This is not an easy task,

and the discovery of the right trade-off can require multiple evaluations of the problem. The advantage EAs have in these situations is that being population based they allow the discovery of multiple possible trade-offs at once.

An important concept when dealing with **Multi-objective optimization (MOO)** problems is the one of *Pareto dominance* [114]. To explain it, consider two candidate solutions for a MOO problem,  $x_1$  and  $x_2$ . The solution  $x_1$  is said to *dominate*  $x_2$  if and only if the two following conditions apply:

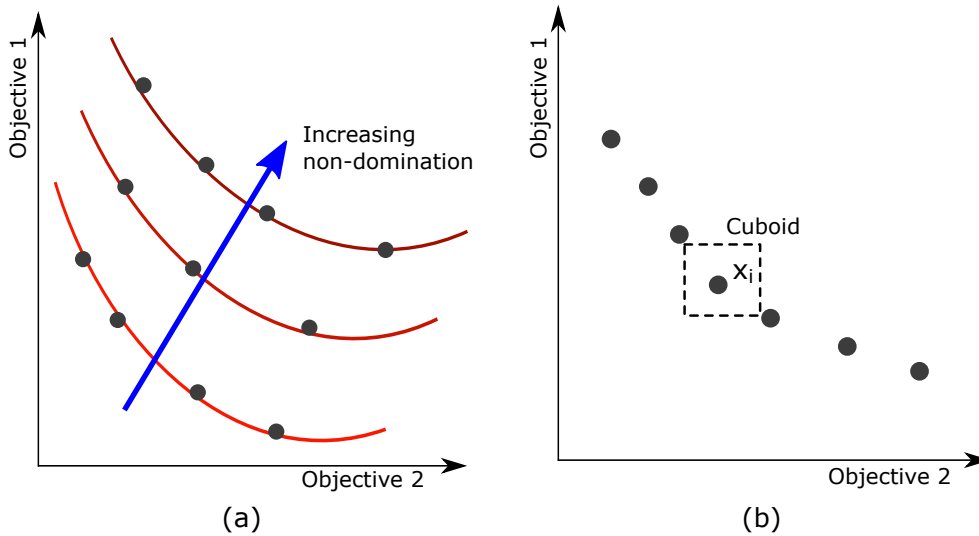
1.  $x_1$  is not worse than  $x_2$  on any of the problem's objectives;
2.  $x_1$  is better than  $x_2$  on at least one of the objective functions.

An example of how a solution  $x_1$  divides the plane between dominated and non-dominated regions is shown in Fig. 2.5.(a). Among all the possible solutions to a MOO problem, the set of *non dominated* solutions contains all the best trade-offs that can be found for the problem. This set is referred as *Pareto front*. A representation of a possible Pareto front for a MOO with two objectives to maximise is shown in red in Fig. 2.5.(b). In blue are represented the solutions belonging to the front, while all the other dominated solutions are shown in grey.



**Figure 2.5:** An example of a MOO with two objectives to maximize. (a) A possible solution  $x_1$  divides the plane defined by the two objectives into multiple areas of domination and non-domination. (b) A possible Pareto front, in red, for a MOO problem.

A well known EA designed to deal with MOO problems is NSGA-II [115]. This method works by sorting all the possible solutions into non-dominated fronts on an ascending level of non-domination, as shown in Fig. 2.6.(a). At each generation  $g$ , the new population  $\Gamma_{g+1}$  is then filled according to front ranking, by first adding to it the elements from the most non-dominated front, then the ones from the second-most non-dominated front, etc., until the population is complete. If a front is only partially selected, that is, if the solutions on the front are more than the remaining positions in  $\Gamma_{g+1}$ , only the ones with the highest *crowding distance* are selected [115, 116]. This distance



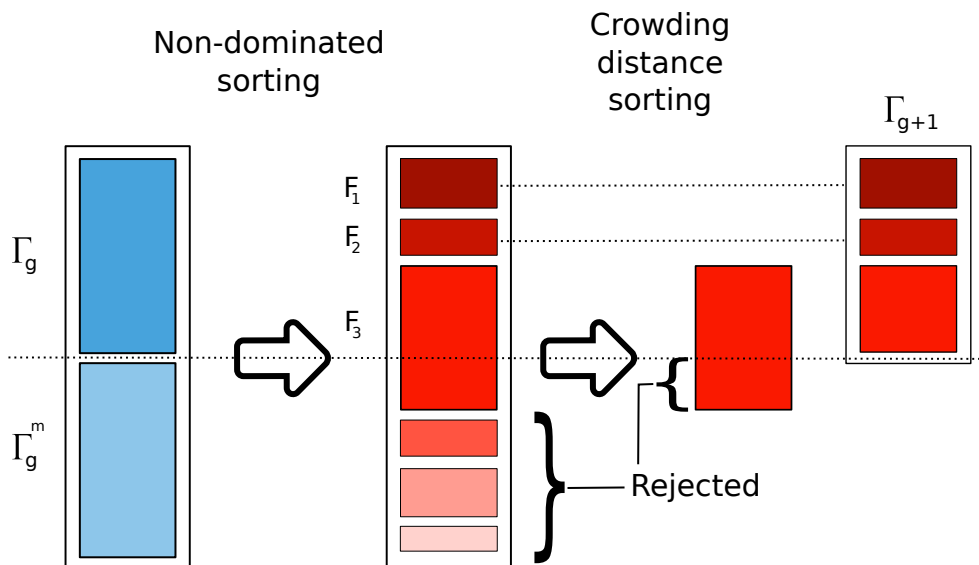
**Figure 2.6:** (a) Representation of non-dominated fronts sorting. Each grey circle represents a solution with respect to the two optimization objectives. The blue arrow represents the direction of increasing non-domination. (b) Representation of crowding distance cuboid calculation around solution  $x_i$ .

is a metric providing an estimate of the density of individuals around any single solution  $x_i$ . To have meaning, it needs to be calculated only between solutions on the same non-dominated front. The calculation is performed by measuring the area of the largest cuboid surrounding an individual  $x_i$ , without including any other solution on the same front, as shown in Fig. 2.6.(b) [116].

An overview of the whole NSGA-II selection procedure is illustrated in Fig. 2.7.

### 2.2.2 Searching for Diversity

EAs require the definition of a fitness function measuring the quality of the solutions with respect to a given goal because they are *objective-based*. The designer has to know in advance the goal to reach and the way the progress towards this goal can be measured. Designing such a fitness function is not an easy task. Moreover, if the function is not properly defined, it can lead the EA to fall victim of deception and converge to local optima [35]. This can lead to a phenomenon known as *population collapse*, in which the whole population is composed by similar individuals, preventing any possible advancement out of the local optimum [32, 33]. There are multiple ways to overcome this kind of problem. An approach is to use additional objectives through a MOO approach [117, 118]. By using the additional objectives, the population can escape local optima with respect to the main optimization objective. Another promising strategy is to force the algorithm to preserve the diversity of the population. In order to do so, many approaches tried to preserve the diversity of the population from the genotype point of view [119]. This can be done through multiple mechanisms as speciation and clustering [110, 120] or tournament



**Figure 2.7:** Schematic of the NSGA-II procedure as shown by [115].  $\Gamma_g$  is the population at generation  $g$ , while  $\Gamma_g^m$  are the corresponding offsprings.  $F_i$  are the non-dominated fronts in which the solutions are divided before selection. Finally  $\Gamma_{g+1}$  is the new population at generation  $g + 1$ .

selections [121]. However, preserving diversity in the genotype space does not imply that the individuals will behave in different ways. Multiple genotypes can express the same kind of phenotype. For this reason, a more interesting approach is to push the algorithm to preserve diversity in the phenotype space. Maximizing the diversity in the phenotype space also allows for the discovery of many different ways in which a problem can be solved. This means that rather than returning a single solution, the algorithm should return a whole set of diverse solutions. A very well known algorithm using this strategy is [Novelty Search \(NS\)](#) [34]. Introduced by Lehman and Stanley, the method performs the search not by targeting a fitness objective, but by searching for a set of individual whose phenotypes are as different as possible. Algorithms performing the search through this strategy can be defined as *divergent search algorithms* [122]. Moreover, this strategy allows the algorithm to not get stuck in any local optima that could possibly be present in the fitness landscape.

Note that, since [NS](#) was introduced in the context of evolutionary robotics and artificial life, the phenotype usually consisted in the way the agent behaved. For this reason [NS](#), and similar methods, are said to maximize the *behavioral diversity* of the individuals [123] and the phenotype itself is referred to as *behavior descriptor*. Considering this, and the fact that the focus of this manuscript is the learning of policies in situations of sparse rewards, from now on we will consider the individuals generated by the presented algorithms as policies  $\pi(\cdot)$  parametrized by a set of parameters  $\theta \in \Theta$ . As stated in [Sec. 2.2](#), this parameters correspond to the genotype of the individuals, while their phenotype corresponds to the behavior these policies have in the environment. Thanks to the flexibility of [EAs](#), this point of view comes with

little loss of generality but it will help in keeping the discussion aligned to the way these methods are presented in the literature.

### Novelty Search

**Novelty Search (NS)** is an **EA** that works by replacing the fitness metric used by standard **EAs** with a *novelty* metric. This metric pushes the search towards novel areas of the search space. The novelty is calculated in a space  $\mathcal{B}$  called **Behaviour Space (BS)** in which the behavior of each policy  $\pi(\cdot)$  parametrized by  $\theta_i \in \Theta$  is represented. This space is usually hand-designed by taking into account the system and the task to be fulfilled. The policies generated by the algorithm are run on the system for a given number of steps  $T$ , traversing a trajectory of states  $\tau_S = [s_0, \dots, s_T]$ , where each  $s_t$  corresponds to the state of the system at time step  $t$ . These traversed states are observed through some sensors, such that they produce a corresponding trajectory of observations  $\tau_O = [o_0, \dots, o_T]$ , with  $o_t \in \mathcal{O}$ , where  $o_t \in \mathcal{O}$  is a potentially under-complete observation of the state of the system at time  $t$ . An observer function  $O_B : \mathcal{O}^T \rightarrow \mathcal{B}$  then maps this trajectory of observations to a hand-designed *behavior descriptor*  $b_i$  for the policy  $\theta_i$ . The overall process can be summarized by introducing a behavior function  $\phi$  mapping each policy  $\theta_i$  to its corresponding behavior descriptor:

$$\phi(\theta_i) = b_i \quad (2.5)$$

The descriptor usually consists in a vector of real numbers. For example, in the case of a robotic arm, it can be the final position and orientation of its end effector.

Once computed, the descriptors are used to calculate the policies' novelty. This novelty represents how different the behavior of each policy is with respect to the behavior of the other agents. In practice it is calculated as the average distance between the behavior descriptor of a given policy  $\theta_i$  and the descriptors of the  $k$  closest policies in the behavior space  $\mathcal{B}$ . It is calculated as:

$$\eta(\theta_i) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(b_i, b_j) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(\phi(\theta_i), \phi(\theta_j)), \quad (2.6)$$

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

The novelty of the policies is calculated at each generation  $g$  and used to choose the policies for the next generation. Moreover,  $N_Q$  policies are sampled to be stored into an *archive*  $\mathcal{A}_{\text{Nov}}$ , or *repertoire*, returned as outcome of the algorithm. These policies can be chosen either randomly or by selecting the most novel ones from the current generation of offsprings [124]. The archive is also used to keep track of the already explored areas of the space  $\mathcal{B}$ . This is done by choosing the  $|J|$  closest neighbors used in equation (2.6) not only from the current population and offspring but also from the archive. The whole **NS** approach is shown in Alg. 1, in which the evaluation budget  $Bud$  is equal to the total number of policy evaluations.

By choosing the most novel policies from the current generation to compose the next population, the search is always pushed towards less explored areas

---

**Algorithm 1: Novelty Search**

---

**INPUT:** evaluation budget  $Bud$ , parameter space  $\Theta$ , behavior space  $\mathcal{B}$ , variation function  $\mathbb{V}(\cdot)$ , population size  $M$ , number of offspring per parent  $m$ , number of policies to add to archive  $N_Q$ ;  
**RESULT:** archive  $\mathcal{A}_{Nov}$ ;  
Initialize  $\mathcal{A}_{Nov} = \emptyset$ ;  
Initialize generation counter  $g = 0$ ;  
 $\Gamma_0 \leftarrow M$  policies from  $\Theta$ ;  
Evaluate  $\theta_i, \forall \theta_i \in \Gamma_0$ ;  
Calculate behavior descriptor  $b_i = \phi(\theta_i) \in \mathcal{B} \forall \theta_i \in \Gamma_0$ ;  
**while**  $Bud$  not depleted **do**  
    Generate offsprings  $\Gamma_g^m \leftarrow \mathbb{V}(\Gamma_g)$ ;  
    Evaluate  $\theta_i, \forall \theta_i \in \Gamma_g^m$ ;  
    Calculate behavior descriptor  $b_i = \phi(\theta_i) \in \mathcal{B} \forall \theta_i \in \Gamma_g^m$ ;  
    Calculate novelty  $\eta(\theta_i) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(b_i, b_j), \forall \theta_i \in \Gamma_g \cup \Gamma_g^m$ ;  
     $\mathcal{A}_{Nov} \leftarrow N_Q$  samples from  $\Gamma_g^m$ ;  
    Generate  $\Gamma_{g+1}$  with most novel  $\theta_i \in \Gamma_g \cup \Gamma_g^m$ ;  
     $g = g + 1$ ;  
**end**

---

of  $\mathcal{B}$ . Doncieux et al. have shown that the repertoire will tend to uniformly cover the BS [125]. At the same time, measuring and comparing behaviors is still an open question, requiring the definition of a good behavior representation. Nonetheless, as long as the BS is properly designed, NS is a very good exploration algorithm in situations in which the reward is either sparse or contains local minima.

Since the introduction of NS, many divergent search algorithms have been developed, using different mechanisms to drive the search: curiosity [126], empowerment [127], surprise [45], diversity [46, 44, 42, 41], and novelty [40].

### Quality-Diversity algorithms

Notwithstanding the capacity for exploration of NS, completely ignoring the fitness function prevents the algorithm to exploit any of the rewards potentially found during the search. This leads the set of solutions returned by the method to have arbitrary performances. Trying to overcome this problem has led to the development of the QD family of algorithms [42, 41]. The methods of this family generate a set of diverse solutions that at the same time have high performances with respect to the given objective.

Among the first QD algorithms developed is **Novelty Search with Local Competition (NSLC)**, an algorithm that combines NS with the mechanism of *niching* to foster the evolution of high performances solutions [40]. The niching mechanism consists in considering policies that are close enough in the BS as belonging to the same niche. The competition with respect to performances

then happens only between members of the same niche. The rationale for this approach is that agents with high performance in an area of the search space cannot dominate the ones in a different area that have lower performances, thus preventing the collapse of diversity during the search process. This is done by calculating a *local competition objective* for each agent  $\theta_i$  by counting how many agents among the  $k$  nearest neighbors used to calculate the novelty in Eq. (2.6) have lower fitness than  $\theta_i$ . The objective encodes the performance of an agent with respect only to its nearest neighbors. The algorithm then uses a MOO strategy to optimize both the novelty of the agents and their local competition objective. This strategy allowed the authors to generate a wide set of robot morphologies capable of efficient locomotion.

NS based methods are characterized by two main factors: they can work with continuous BS, and the population of agents performing the search is kept separated from the archive of stored solutions. A different approach is the one introduced by Mouret and Clune with MAP-Elites (ME) [43]. This method works by discretizing its BS through a multidimensional grid in which the agents are organized. Moreover, the algorithm does not keep the population separated from the archive: the agents in the grid act both as population and as archive. The method works by randomly initializing a set of policies  $\theta$  that are evaluated in order to calculate their behavior descriptor. The policies are then placed in the cells of the BS grid corresponding to their descriptor. At this point, the main cycle of ME starts. The algorithm samples one of the policies from the grid and generates a new agent by mutating the sampled individual. The new policy is then evaluated and its behavior descriptor  $b_i$  calculated. If the cell of the grid in which the policy belongs thanks to  $b_i$  is empty, the agent is placed in it and another element from the grid is sampled. On the contrary, if the cell is already occupied by another policy, the algorithm performs a tournament selection between the two policies, only storing the one with the highest fitness. The cycle is repeated until the whole evaluation budget is depleted. With time, this allows to improve on the quality of the discovered solutions. The whole process is shown in Alg. 2.

The discretization of the search space allows ME to reduce its memory and computational footprints with respect to the continuously growing archive of NS. The cell look-up operation in ME has a computation cost of just  $O(1)$ . At the same time, the computational cost of the nearest neighbors calculation in NS is  $O(n \log n)$  [128], increasing at each generation with the size of the archive. Thanks to its performances and its simplicity, ME has become one of the most well known and widely used QD approaches.

Discovering multiple policies and collecting them into an archive, as done by QD methods, allows for great generalization, even to settings not seen at training time. An example of this is the work done by Cully et al. in which a six legged robot learned how to recover from damage received to one of its legs and still be able to walk [46]. Initially, the method generates a collection of policies that allowed the undamaged robot to walk. After one of the robot’s legs is disabled, the algorithm explores its collection of policies in order to find the most promising solution that still allows the robot to advance. This solution

---

**Algorithm 2: MAP-Elites**

---

**INPUT:** evaluation budget  $Bud$ , parameter space  $\Theta$ , discretized behavior space  $\mathcal{B}$ , variation function  $\mathbb{V}(\cdot)$ , number of initial policies  $M$ ;

**RESULT:** discretized archive  $\mathcal{A}_{ME}$ ;

Initialize  $\mathcal{A}_{ME} = \emptyset$ ;

$\Gamma_0 \leftarrow M$  policies from  $\Theta$ ;

Evaluate  $\theta_i, \forall \theta_i \in \Gamma_0$ ;

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

$\mathcal{A}_{ME} \leftarrow \theta_i, \forall \theta_i \in \Gamma_0$ ;

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

    Sample  $\theta_i$  from  $\mathcal{A}_{ME}$ ;

    Generate offspring  $\tilde{\theta}_i \leftarrow \mathbb{V}(\theta_i)$ ;

    Evaluate  $\tilde{\theta}_i$ ;

    Calculate behavior descriptor  $b_i = \phi(\tilde{\theta}_i) \in \mathcal{B}$ ;

**if**  $\mathcal{A}_{ME}(b_i) == \emptyset$  **then**

$\mathcal{A}_{ME}(b_i) \leftarrow \tilde{\theta}_i$ ;

**else**

$\theta_j \leftarrow \mathcal{A}_{ME}(b_i)$ ;

**if**  $fitness(\theta_j) < fitness(\tilde{\theta}_i)$  **then**

$\mathcal{A}_{ME}(b_i) = \emptyset$ ;

$\mathcal{A}_{ME}(b_i) \leftarrow \tilde{\theta}_i$ ;

**end**

**end**

**end**

---

is then evaluated and its performance on the damaged robot calculated. The algorithm then updates the performances not only of the selected policy, but also of its neighbors, in order to quickly reduce its uncertainty about the quality of all solutions with respect to the new situation. The rationale of this is that similar behaviors have similar performances even in the new setting. This reduced uncertainty allows the algorithm to better select which policy to test next. The cycle repeats until performance on the damaged robot is 90% or greater of the maximum expected performance for any behavior [46].

**ME** has been extended in **CVT-ME** to address the explosion in the number of cells of the grid while working in high-dimensional **BS** [129]. In situations of high-dimensional **BS**, the number of cells increases exponentially with the number of dimensions. With 50 cells per dimension, in the case of a 2-dimensional **BS**, the total number of cells will be  $50 \times 50 = 2500$ . This numbers increases to  $50 \times 50 \times 50 = 125000$  in a space with just 3 dimensions. The exponential increase in dimensions reduces the performances of **ME** due to the fact that the occupancy of the cells will be very sparse, so very little performances optimization will be performed. This explosion can be reduced by autonomously changing the size and number of cells along each dimensions



[129].

Other extensions address the issue of working in noisy domains. In order to properly compare the behaviors of the policies, QD algorithms need to work in deterministic settings. This greatly limits the range of applicability of these methods. The problem can be addressed in multiple ways. Justensen et al. used adaptive sampling and drifting elites [130] to strengthen the resistance of ME to noise. The method consists in the multiple evaluation of the policies in order to estimate their behaviors and performances. An agent is evaluated multiple times until it either: it ends up in an empty cell or, if it remains in an already filled cell, it has been evaluated as many times as the best performing policy in that cell, the *elite*. If the mean performance is higher than the one of the elite, the agent is added to the grid, otherwise it is discarded. Moreover, the number of evaluations performed on each individuals is increased with time, in order to refine their performance estimate. To prevent cells of the grid to become empty once the reevaluation of a policy makes it move from one cell to the other, multiple solutions are stored in the same cell. From all the solutions in a single cell, only the best performing ones will be considered for reproduction when the cell is selected to generate a new policy. Another approach storing multiple agents in the same ME cell to deal with the problem of noisy domains is *deep-grids* [131]. Contrary to what has been done by Justensen et al., deep grids does not use only the elite from a selected cell to generate new offsprings, but rather samples the agents according to their fitness. This allows to generate a collection of policies that is more stable to noise.

QD algorithms have also been combined with other kind of EAs, namely Evolution Strategies (ES), to increase their efficiency and speed of convergence. Evolution Strategies (ES) is a family of fitness based EAs that work by estimating a distribution from which the population to be evaluated is sampled [132]. After each evaluation, the distribution is updated according to the performances of the sampled agents and a new population is sampled. ESs are particularly efficient EAs and have been recently shown to perform comparably to RL methods in certain situations [133].

Conti et al. introduced an ES that used NS's novelty objective to look for novel solutions while improving their performances. This allows to improve the ES exploration in setting of sparse rewards while retaining scalability. The novelty metric allowed the method to overcome the local optima in which standard ES would easily get stuck. A different approach is the one of *emitters* [55, 56]. These works use ME as a scheduler to launch instances of ES that perform local search in the BS around their point of initialization. The search is then performed while optimizing the quality of the discovered solutions. These instances of ES performing search in a neighborhood of the search space are called emitters. The concept of emitters is very flexible, allowing any reward-based method to be used as an emitter in combination with a QD algorithm.

QD algorithms have also been combined with RL methods [134, 135], by using the EA to collect the data on which the RL algorithm is trained. This

allows for better exploration by overcoming any possible deceptive gradients that could undermine the performances of gradient-based approaches. At the same time, the data efficiency of the [RL](#) approach allows to quickly optimize the policy with respect to the goal. The higher-performing gradient-optimized policy is then injected into the evolving population to steer the search towards more profitable areas.

To conclude, the thesis focuses on the study and development of algorithms capable of dealing with sparse reward settings. In these situations, a good strategy for the agent is to focus on exploration until a reward is discovered, at which point the agent has to be able to quickly optimize the rewarding solution. While [RL](#) methods can efficiently optimize policies thanks to gradient descent, they struggle with hard to explore problems. On the contrary, [QD](#) methods have proven to be very powerful in solving problems in which exploration is difficult. Nonetheless, they present one major limitation: diversity is measured in low-dimensional [BS](#) that are usually hand-designed. This requires more involvement from the system's designer, leading to the possible introduction of bias and to increased costs of deployment. The next chapter will present an approach designed to deal with this limitation.



**Chapter content**

---

<b>3.1</b>	<b>Introduction</b>	<b>31</b>
<b>3.2</b>	<b>Related work</b>	<b>32</b>
<b>3.3</b>	<b>Methodology</b>	<b>34</b>
3.3.1	Policy selection	36
3.3.2	Search and Training	38
<b>3.4</b>	<b>Experiments</b>	<b>38</b>
3.4.1	Experimental setup	38
3.4.2	Evaluation	40
3.4.3	Results	42
<b>3.5</b>	<b>Conclusion</b>	<b>45</b>

---

This chapter is adapted from the following publication:

*Paolo, G., Laflaquiere, A., Coninx, A., & Doncieux, S.* **Unsupervised learning and exploration of reachable outcome space.** In 2020 IEEE International Conference on Robotics and Automation (ICRA 2020).

---

**3.1 Introduction**

Chapter 2 discussed how, in order to deal with sparse rewards, a good strategy is to completely focus on exploration. Divergent search EAs like QD methods are good candidates for this. This is due to multiple reasons. First, these methods generate simple policies, each one specialized in reaching a sub-part of the search space, rather than looking for a single complex policy able to cover the whole space as is the case for RL algorithms. This can seem limiting from the point of view of generalization: being so simple, these policies cannot generalize to new problems. However, the whole collection of policies returned by QD methods can be used on different problems, as long as the collected policies are reevaluated to assess their performances. This strategy has been useful, for instance, in making a robot resilient to damage [46] or to generate complex behaviors by combining these simple policies in the context of hierarchical RL [44]. Moreover, these methods do not need any reward to

drive the exploration and find new policies, the search is therefore not misled by deceiving reward gradients. At the same time, an outcome space can be shared by different tasks and different domains, meaning that the same repertoire of policies can thus be applied to multiple tasks a posteriori [36, 37, 38, 39].

This chapter will present one of these divergent search algorithms: the **Task Agnostic eXploration of Outcome space through Novelty and Surprise (TAXONS)** method [136]. It takes advantage of the exploration strength of **NS** while overcoming one of its main limitations: the need to hand-design the outcome space, also called behavior space, in which the novelty of each policy is evaluated. **NS** has recently been shown to tend towards a uniform exploration of the outcome space, which is an unbiased strategy in the absence of reward [125]. Removing the need to hand-design the **BS** helps in reducing the amount of prior information needed at design time, rendering the algorithm more widely applicable. The outcome space, in fact, needs to be adapted for any new agent and/or environment. Apart from being costly in terms of human resources, defining by hand the appropriate features of the outcome space requires for the experiment designer to know the features of the robot, environment, and tasks. The search will also be constrained by the biases of the designer's choices. Being able to autonomously learn the outcome space in which policies are discriminated can remove said limitations while improving the applicability of these approaches. **TAXONS** does so by learning the outcome space while performing the search through the use of an **autoencoder (AE)** [49]. **AEs** are a family of **NNs** architectures commonly used for dimensionality reduction.

In the following, other approaches from the literature that learn the **BS** will be presented in section 3.2, then **TAXONS** will be introduced in section 3.3. Section 3.4 will discuss the experiments performed to test the method and its related results. The chapter will conclude with section 3.5 with a recap of what has been done and a discussion on the next steps and ideas explored in the rest of the thesis.

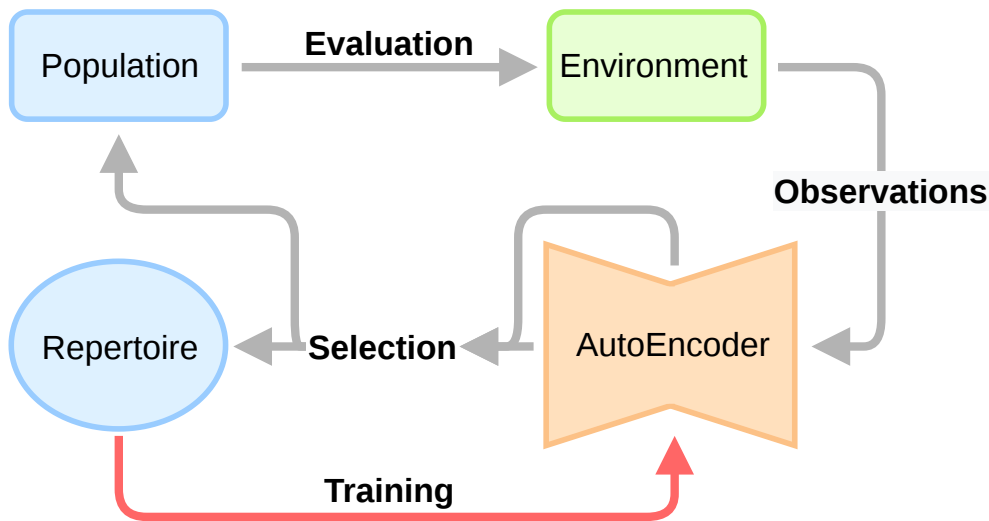
## 3.2 Related work

Given the important role played by the **BS** in divergent search algorithms, its design is fundamental. Hand-designing it can be problematic, requiring prior knowledge on the system and on the problem to be solved. Moreover, the designer can introduce some unexpected bias in situations in which it is not clear what features would benefit the search. For these reasons, methods combining representation learning algorithms with **EAs** have recently been proposed. Representation learning algorithms are a family of methods that can learn abstract features characterizing data. They can be used to autonomously represent high-dimensional data into a lower dimensional space [137]. Combining them with **EAs** reduces the amount of task related prior knowledge required at design time, moving the engineering efforts from the design of the task spe-

cific representation to the design of the task agnostic representation learning algorithm. This makes the approaches more flexible with respect to the kind of problem they are applied on: the same representation learning method can learn good representations for different situations without many algorithmic changes.

Meyerson et al. [138] proposed to learn a domain-specific behavior descriptor by starting from an underlying, generic descriptor requiring minimal domain-specific knowledge. The domain-specific descriptor then is in the form of a weighted vector over all the possible state-action pairs, thus reducing the need to take into account every situation at design time. At the same time, having to weight every state-action pair limits the approach to discrete and low-dimensional domains. In DeLeNoX[139], the authors developed a NS-based algorithm to evolve 2-dimensional space-ships shapes in which the novelty of each shape was calculated in the low-dimensional feature space of an AE. The fact that the AE is trained from scratch on each new generation of shapes is the limiting factor of the algorithm, removing any memory of previous generations.

Autonomous learning has not been limited only to algorithms relying on continuous BS like NS, but has also been applied to approaches with discrete ones like MAP-Elites[46]. Innovation Engines [140] generate and classify novel images thanks to the learned feature space of an AE. However, the AE is trained beforehand on a dataset of images and not during the search process. More recently, the approach introduced by Cully and Demiris [141] uses a hierarchical strategy to combine hand-designed features with learned ones. Having a hierarchy of BSs has the advantage of requiring the update of only a part of the BSs stack when changing the task, rendering the algorithm more flexible. Another approach that uses an AE to learn the BS online while searching for policies is AURORA [142]. In this work the AE learns the space from the trajectories of the raw observations of the internal states of the system. The learned space is then used to differentiate the policies by calculating the distances among the generated trajectories projected in the learned space. Notwithstanding the power of the algorithm to discover a host of different trajectories, using observations of the internal states can be limiting in situations in which the states do not carry enough information. For example if a robot is being trained to move a box from one position to another, the box needs to be explicitly tracked in order to distinguish between the different outcomes of the policies, adding another layer of complexity to the algorithm. It is to notice that while AURORA is similar in spirit to TAXONS, there are some important differences. Namely, TAXONS learns the search space directly from high-dimensional RGB images of the environment, extracting the features in an unsupervised way. Moreover, the diversity of each policy is assessed not only by using the self-learned outcome space, but also by taking advantage of the information provided by the reconstruction error of the AE. This added measure helps improve both the quality of the search space and the diversity of the generated policies. At the same time, AURORA can work on whole trajectories while TAXONS can only work on the last observation.



**Figure 3.1:** High level schematic of *TAXONS*. It consists of two processes operating in parallel. The first one, the search process (gray arrows), generates a set of new policies, evaluates them in the environment and stores the best ones in the repertoire. The second process, highlighted by the red arrow, is the training of the AE on the observations collected during the evaluation of the policies.

These features will be discussed in more detail in the next section.

### 3.3 Methodology

As already mentioned, hand-designing the *BS* requires a lot of involvement from the system’s designer. Having to know exactly what he wants to measure, what the task to be solved is, and how to measure the progress towards finding a solution. All of this requires prior knowledge that has to be collected by accurately studying the system.

This problem is approached with *TAXONS*[48], an algorithm based on *NS*. Instead of relying on a hand-designed *BS*, *TAXONS* builds it in the form of a low-dimensional representation of the trajectory of observations  $[o_0, \dots, o_T]$  collected during the evaluation of each policy. Here each observation  $o_t \in \mathcal{O}$  corresponds to the observations defined in section 2.2.2. To simplify the approach, the last observation  $o_T$  of the trajectory is considered to be informative enough to characterize the behaviour of the system during the execution of the related policy. Consequently, only this last observation will be used to build the *BS* representing the outcome of the policy, for this reason also called *outcome space*.

The observations are considered to be high-dimensional RGB images. Working directly on such high-dimensional observations can be complicated, if not infeasible, due to two main reasons:

- The euclidean distance metric, used to calculate the novelty of the be-

haviors, does not work properly in high-dimensional spaces [143];

- In the case of RGB images, calculating the pixel-wise distance would return a distance between the images themselves rather than a distance between the objects in the images. This makes it difficult to distinguish between different situations. An example of this can be seen in figure 3.2. The pixel-wise distance between the second and third images with respect to the first one is the same, notwithstanding the real distance between the disks being different.

For these reasons a dimensionality reduction algorithm is needed to move to a low-dimensional space where a more meaningful distance between observations can be calculated. Among the many dimensionality reduction algorithms [144] that can be used to build the low-dimensional space, this work uses an AE [49], given the power and flexibility these methods have.

AEs are a class of neural network composed by an encoder  $E$  and a decoder  $D$ . The encoder projects its input  $x \in \mathcal{O}$  in a space called *feature space*  $\mathcal{F}$ , usually of lower dimension than the input space. The decoder then returns the projection from the feature space to the output space. The goal of the AE is to learn to reconstruct its input  $x$  on its output:

$$\begin{aligned} E : \mathcal{O} &\rightarrow \mathcal{F}, \\ D : \mathcal{F} &\rightarrow \mathcal{O}. \end{aligned} \tag{3.1}$$

An important point of the process is the fact that the AE has to project  $x$  in the lower-dimensional feature space. This forces it to extract only the important features from the input to properly reconstruct it on its output. This whole process happens in an unsupervised way, with the AE being trained by minimizing the error function

$$L(x) = ||x - D(E(x))||^2. \tag{3.2}$$

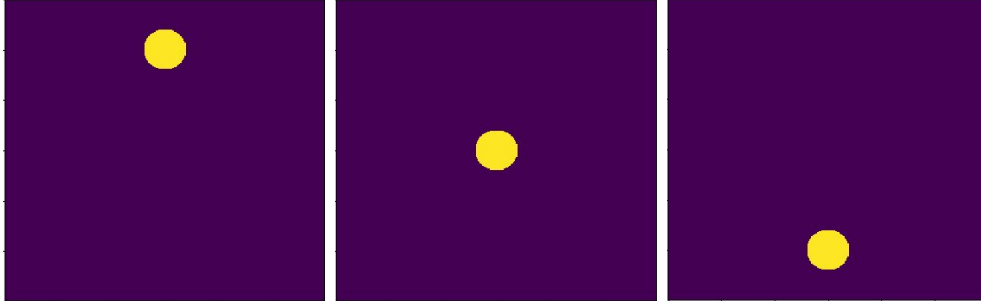
The whole AE is trained in an online fashion on the observations generated at each iteration when evaluating the new policies.

TAXONS uses the encoder  $E$  as observation function, to extract the features from the image, and its lower dimensional feature space  $\mathcal{F}$  as outcome space. Note that in this process no task or reward is required, thus the algorithm needs minimal interventions from the designer, apart from choosing which observation to feed to the AE.

A high-level schematic of TAXONS is shown in figure 3.1, while a more detailed view is given in Alg. 3. The method consists of two processes running in parallel:

- **Search process:** highlighted in gray in figure 3.1, it generates and evaluates the policies, storing the ones with higher diversity in the repertoire. It does so by firstly generating a population of policies, evaluating them in the environment, and then selecting the most novel ones to add to the repertoire. During the evaluation, the last observation  $o_T$  from each of





**Figure 3.2:** *The pixel-wise distance between the second and third images with respect to the first one is the same, notwithstanding the real distance of the circle in the euclidean space being different.*

the policies is collected and fed to the **AE** that will be used to extract the features for the selection step. This last step is fundamental to the correct operation of **TAXONS** and is described in detail in section 3.3.1;

- **AE training:** highlighted by the red arrow in figure 3.1, is the training of the **AE** on the observations collected during the evaluation of the policies. This training process happens in parallel with the search process and is described in more detail in section 3.3.2.

### 3.3.1 Policy selection

While in **NS** only the distance in the **BS** is used as a metric, in our approach the best policies are selected according to two metrics, ensuring both their novelty and the representativity of the behaviour space. The first one, referred to as *novelty*, corresponds to the novelty metric of **NS** already defined in Eq. (2.6), reported here for clarity:

$$\eta(\theta_i) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(b_i, b_j) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(\phi(\theta_i), \phi(\theta_j)), \quad (3.3)$$

More precisely, the mapping  $\phi$  in (2.5) is replaced by the mapping:

$$f(\theta_i) = E(o_T^{(\theta_i)}), \quad (3.4)$$

where  $o_T^{(\theta_i)}$  is the last observation generated by the policy  $\theta_i$ . This allows us to rewrite Eq. (2.6) as:

$$\eta(\theta_i) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(f(\theta_i), f(\theta_j)) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(E(o_T^{(\theta_i)}), E(o_T^{(\theta_j)})) \quad (3.5)$$

The second metric, referred to as *surprise*, corresponds to the reconstruction error of the **AE**; it is expressed as:

$$s(\theta_i) = \|o_T^{(\theta_i)} - D(E(o_T^{(\theta_i)}))\|^2. \quad (3.6)$$

This reconstruction error tends to be large when the **AE** processes observations which have not been frequently encountered yet. This idea is similar to the one introduced with the **RND** approach [87]. Maximising this metric during the search pushes new policies to explore novel parts of the state (observation) space. This ensures that the observations are representative of the states the system can reach. In practice, one of the two metrics is picked with a probability of 0.5 to evaluate every new iteration of policies. This strategy is similar to the one used by Doncieux and Mouret [113] to mix different behaviour descriptors.

Combining these two metrics drives the search towards learning an outcome space that is representative of the reachable states of the system and towards policies that are diverse in this space.

---

**Algorithm 3: TAXONS**

---

**INPUT:** evaluation budget  $Bud$ , parameter space  $\Theta$ , training interval  $I$ , variation function  $\mathbb{V}(\cdot)$ , population size  $M$ , number of offsprings per parent  $m$ , number of policies to add to archive  $N_Q$ ;  
**RESULT:** archive  $\mathcal{A}_{Nov}$ ;  
Initialize **AE**  $D(E(\cdot))$  with random parameters;  
Initialize population  $\Gamma_0 \leftarrow M$  policies from  $\Theta$ ;  
Initialize  $\mathcal{A}_{Nov} = \emptyset$ ;  
Initialize dataset buffer  $DS = \emptyset$ ;  
**while**  $Bud$  not depleted **do**  
    Generate offsprings  $\Gamma_g^m \leftarrow \mathbb{V}(\Gamma_g)$ ;  
    **for**  $\theta_i \in \Gamma_g \cup \Gamma_g^m$  **do**  
        Evaluate policy  $\pi(\theta_i) \rightarrow o_T$ ;  
        Calculate outcome descriptor  $E(o_T^{(\theta_i)}) = b_i$ ;  
        Calculate surprise  $s(\theta_i) = \|o_T^{(\theta_i)} - D(E(o_T^{(\theta_i)}))\|^2$ .;  
        Store outcome observation  $o_T \rightarrow DS$ ;  
    **end**  
    Calculate novelty  $\eta(\theta_i) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(b_i, b_j)$ ,  $\forall \theta_i \in \Gamma_g \cup \Gamma_g^m$ ;  
     $\mathcal{A}_{Nov} \leftarrow N_Q$  most novel from  $\Gamma_g^m$ ;  
    Substitute  $N_Q$  less novel  $\theta_i \in \Gamma_g$  with  $N_Q$  most novel in  $\theta_i \in \Gamma_g^m$ ;  
    **if**  $g$  multiple of  $I$  **then**  
        Train  $D(E(\cdot)) \leftarrow DS$ ;  
         $DS \leftarrow \emptyset$ ;  
        Update stored policies' descriptors  $b_i$ ,  $\forall \theta_i \in \Gamma_g \cup \mathcal{A}_{Nov}$ ;  
    **end**  
     $g = g + 1$ ;  
**end**

---

### 3.3.2 Search and Training

Similarly to NS, the repertoire of diverse policies is built iteratively. At each iteration, a set of  $M$  new policies, parametrized by  $\theta \in \Theta$ , is generated by modifying the ones from the previous iteration. More precisely, the  $Q$  best policies, according to the metric (novelty or surprise), are duplicated to replace the  $Q$  worst ones. Then the parameters  $\theta$  of all  $M$  policies are perturbed by adding gaussian noise with probability  $p_d$ . Moreover, in the process, the  $Q$  best policies  $\theta_i$  are also stored in the repertoire, along with their final observation  $o_T^{(\theta_i)}$ .

The AE is trained to minimize the reconstruction error by feeding it the observations generated during the policies evaluation. In particular, the final observations  $o_T$  are stored for  $I$  iterations (for a total of  $M \times I$  observations) before the AE is trained for  $J$  epochs. This buffering step helps in collecting enough data for the training process of the AE.

Note that, because the outcome space changes during the training of the AE, the policies in the repertoire are reassigned an updated outcome descriptor after each training episode of the AE, obtained by feeding the associated final observation to the current version of the AE’s encoder.

The TAXONS search process is described in detail in algorithm 3.

## 3.4 Experiments

### 3.4.1 Experimental setup

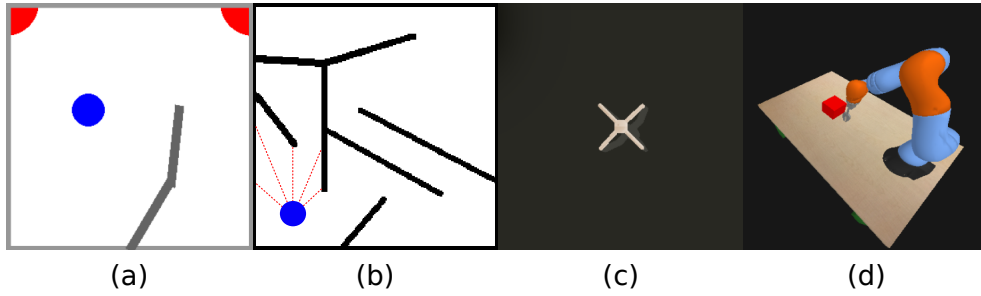
To test if TAXONS can drive exploration by using only high-dimensional RGB images, it has been tested on four different environments:

- **Billiard**: a 2-jointed arm pushing a ball in a 2D room, shown in figure 3.3.(a);
- **Maze**: a two wheeled robot navigating a 2D maze [34], shown in figure 3.3.(b);
- **Ant**: a four legged robot ant moving on the floor [145], represented in figure 3.3.(c);
- **Kuka**: a 7-jointed Kuka robotic arm, simulated in Pybullet [146], that has to learn how to push a box on a table. The setup is shown in figure 3.3.(d).

In all of these situations, driving exploration through images removes the need to track the objects in order to extract the internal states used by NS to drive the search, e.g. the ball or the robot position.

The scenarios are observed through an RGB-camera looking at the scene from the top, with the exception of the Kuka environment that is observed from the side.

TAXONS is compared against five different baselines:



**Figure 3.3:** *The four different experimental environments.*

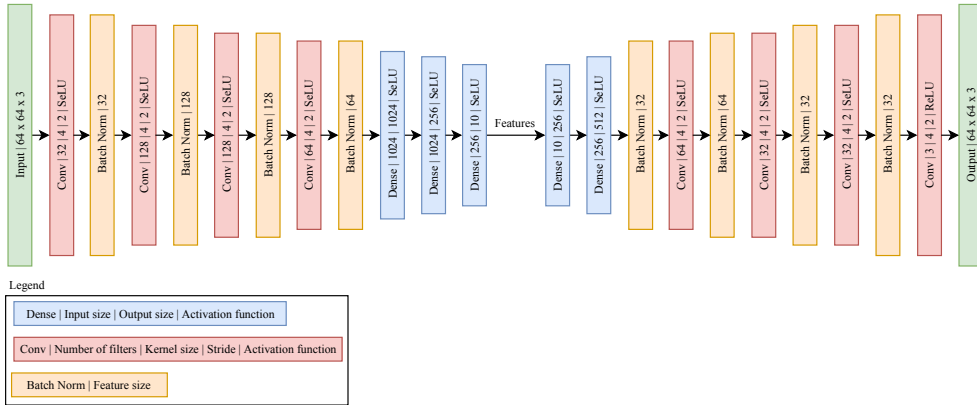
- **NS:** a vanilla novelty search algorithm [34]. This algorithm works with features that are hand-crafted using a priori knowledge about the agent and environment;
- **PNS:** a novelty search algorithm, where the outcome space directly corresponds to the parameter space  $\Theta$  of the policies. The outcome descriptor characterizes the policy but not the final observation. The idea behind this baseline is to verify if diversity in the policy space reflects in diversity in the behavior space;
- **RNS:** a novelty search algorithm where the outcome description of each policy is randomly sampled in a 10D space. The outcome descriptor does not characterize the observation nor the policy. The reason for using this baseline is to verify how important an accurate representation of the behavior is when performing exploration thorough **NS**;
- **RS:** a random search in which all policies are randomly generated and randomly selected to be added to the repertoire. The idea behind the selection of this baseline is to analyze if a random search algorithm can compete with the exploration abilities of **NS** based algorithms;
- **NT:** a novelty search algorithm in which the outcome space corresponds to the feature space of a **AE** whose weights are randomly generated at the beginning of the search and left unmodified during the whole search process. As with **TAXONS**, the **AE** is fed only with the last observation  $o_T$  of the environment. This baseline has been chosen to test the importance of the training process of the **AE**.

The vanilla version of **TAXONS** is also compared against two ablated versions:

- **TAXO-N:** in which only the novelty calculated in the learned feature space is used as selection metric;
- **TAXO-S:** in which only the reconstruction error of the **AE** is used as selection metric.

All experiments have been conducted on a population of  $M = 100$  policies for each iteration. The novelty of each policy is calculated by using a value

of  $k = 15$  neighbours in Eq.(2.6), as proposed by Gomes et al. [124], with the  $N_Q = 5$  best policies added to the repertoire. At each iteration, the parameters  $\theta_i$  of each policy are independently perturbed, with probability  $p_d = 0.2$ , by adding noise sampled from  $\mathcal{N}(0, 0.05)$ . The observations  $o_T$  consist of RGB images of size  $64 \times 64 \times 3$ . The AE consists in an encoder  $E$  with 4 convolutional layers, of sizes  $[32, 128, 128, 64]$  and 3 fully connected layers, of sizes  $[1024, 256, 10]$ ; followed by a decoder  $D$  with 2 fully connected layers, of sizes  $[256, 512]$ , and 4 convolutional layers of sizes  $[64, 32, 32, 3]$ . For the convolutional operations, the kernel is of size 4 and a the stride of size 2 with padding of 1. Moreover, in order to improve the network performances a batch normalization operation [147] is applied after each convolutional layer. The activation functions used are SeLU [148] for every layer, except for the last layer of the decoder, in which a ReLU activation is used to force the non-negativity of the output values. The structure of the AE is shown in figure 3.4. Note that the decoder is smaller than the encoder in order to prevent it from generalizing too well. By having a decoder that is too powerful, after few training iterations, the reconstruction error will be small also for images that have not been seen yet. This would reduce the effect of the surprise metric. The training is done every  $I = 30$  search iterations for  $J = 5$  epochs, with a learning rate of 0.001. The optimizer used is Adam[149].



**Figure 3.4:** AE structure. The input and output of the network are represented in green; in red are the convolutional layers, with the associated batch normalization operation highlighted in orange. In blue are the fully connected layers.

### 3.4.2 Evaluation

The goal of TAXONS is to produce diverse policies. In light of this, the algorithms are compared based on how well they cover the ground-truth outcome space of the system. By design, this ground-truth outcome space corresponds to the  $(x, y)$  position of the ball for the Billiard environment, of the center of the robot for the Maze and Ant environment, and of the box on the table for the Kuka arm environment. The *coverage* metric is thus defined as the percentage of this  $(x, y)$  space reached by the final repertoire of policies. This

is done by dividing this space in a  $50 \times 50$  grid and then calculating the ratio of number of cells reached at least once over the total number of cells.

Note that the ground-truth  $(x, y)$  space is unknown to the methods (except for **NS**) and is only used a posteriori to compare them.

Moreover, to evaluate the statistical significance of the results, each experiment was run 20 times on different random seeds, and the results compared by performing a Mann-Whitney test [150], with Holm-Bonferroni correction [151].

The evolution of the coverage over the training for the different methods is displayed in Fig. 3.6.(a) and the final coverage comparison is displayed in Fig. 3.6.(b).

### Billiard environment

As illustrated in Fig. 3.3.(a), the agent consists in a two-jointed arm, depicted in gray, that can push a blue ball inside a squared room. Two additional corners are depicted in red; in the absence of a task they have no specific function in the simulation. The policy controlling the speed of each joint of the agent is defined by a fifth-degree polynomial **Dynamic Movement Primitive (DMP)** [152]. The policy is run for a time horizon of 500 steps. In order to remove any possible distractor [153], the arm is brought out of the experimental table at the end of the evaluation of each policy. As shown in Fig. 3.5.(b), the final observation  $o_T$  consists in a top-view of the experimental table. Note that, for the **NS** baseline the  $(x, y)$  ground-truth position of the ball is used as outcome descriptor.

The search methods are run for 2000 evaluations.

### Maze environment

As illustrated in Fig. 3.3.(b), the agent consists in a two-wheeled robot, depicted in blue, navigating in a maze [34]. The agent is equipped with 5 distance sensors in the front, represented by the dotted lines. The policy controlling the speed of each wheel of the agent is defined by a 2-layers, fully connected, neural network that takes as input the robot sensors readings. The policy is run for a time horizon of 2000 steps. As shown in Fig. 3.5.(b), the final observation  $o_T$  consists in a top-view of the maze and the agent. Note that, for the **NS** baseline the  $(x, y)$  ground-truth position of the robot is used as outcome descriptor.

The search methods are run for 1000 evaluations.

### Ant environment

As illustrated in Fig. 3.3.(c), the agent consists in a four-legged ant robot [154], moving in a 2D plane of size  $3m \times 3m$ . The policy controlling the torque of each of the 8 agent joints is defined by a sinusoidal **DMP**. The experiment is run for a time horizon of 500 steps or until the robot reaches the borders of

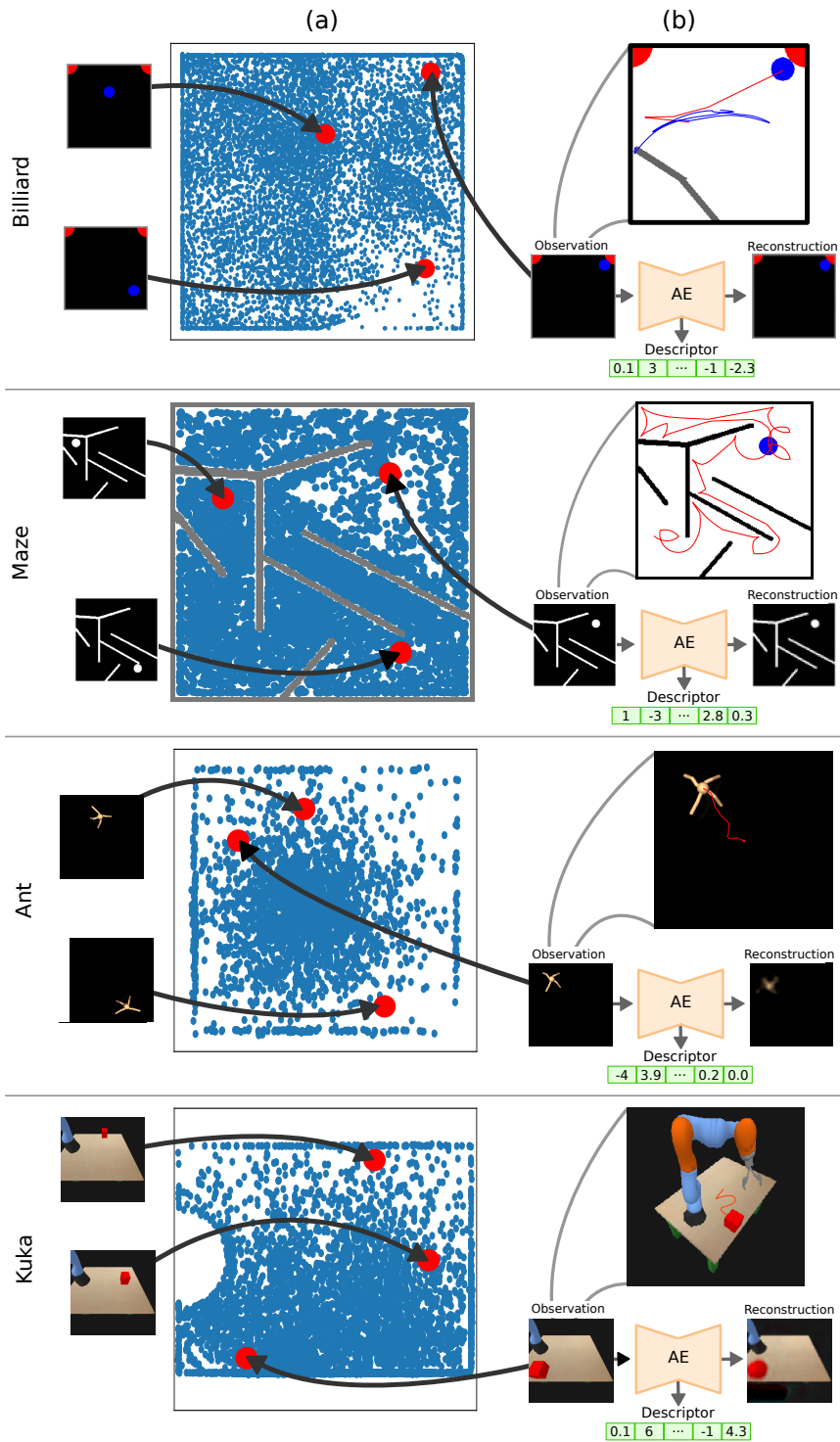
the plane. As shown in Fig. 3.5.(b), the final observation  $o_T$  consists in a top-view of environment. Note that, for the NS baseline the  $(x, y)$  ground-truth position of the robot is used as outcome descriptor. The search methods are run for 500 evaluations.

### Kuka environment

As illustrated in Fig. 3.3.(d), the agent consists in a 7-jointed Kuka robotic arm simulated in PyBullet [146]. The arm can move thanks to a joint position controller and can push a red cube on a table. The policy consists of a sequence of 5 points in the 7-dimensional joint space that need to be reached in a time interval of 2000 timesteps. Moreover, at the end of the policy execution the arm is reset to the initial position to prevent any obstruction of the view of the cube and to reduce the factors of variations in the observation. As shown in Fig. 3.5.(b), the final observation  $o_T$  consists on a lateral view of the table on which the cube rests. Note that, for the NS baseline the  $(x, y)$  ground-truth position of the cube on the table is used as outcome descriptor. The search methods are run for 1000 evaluations.

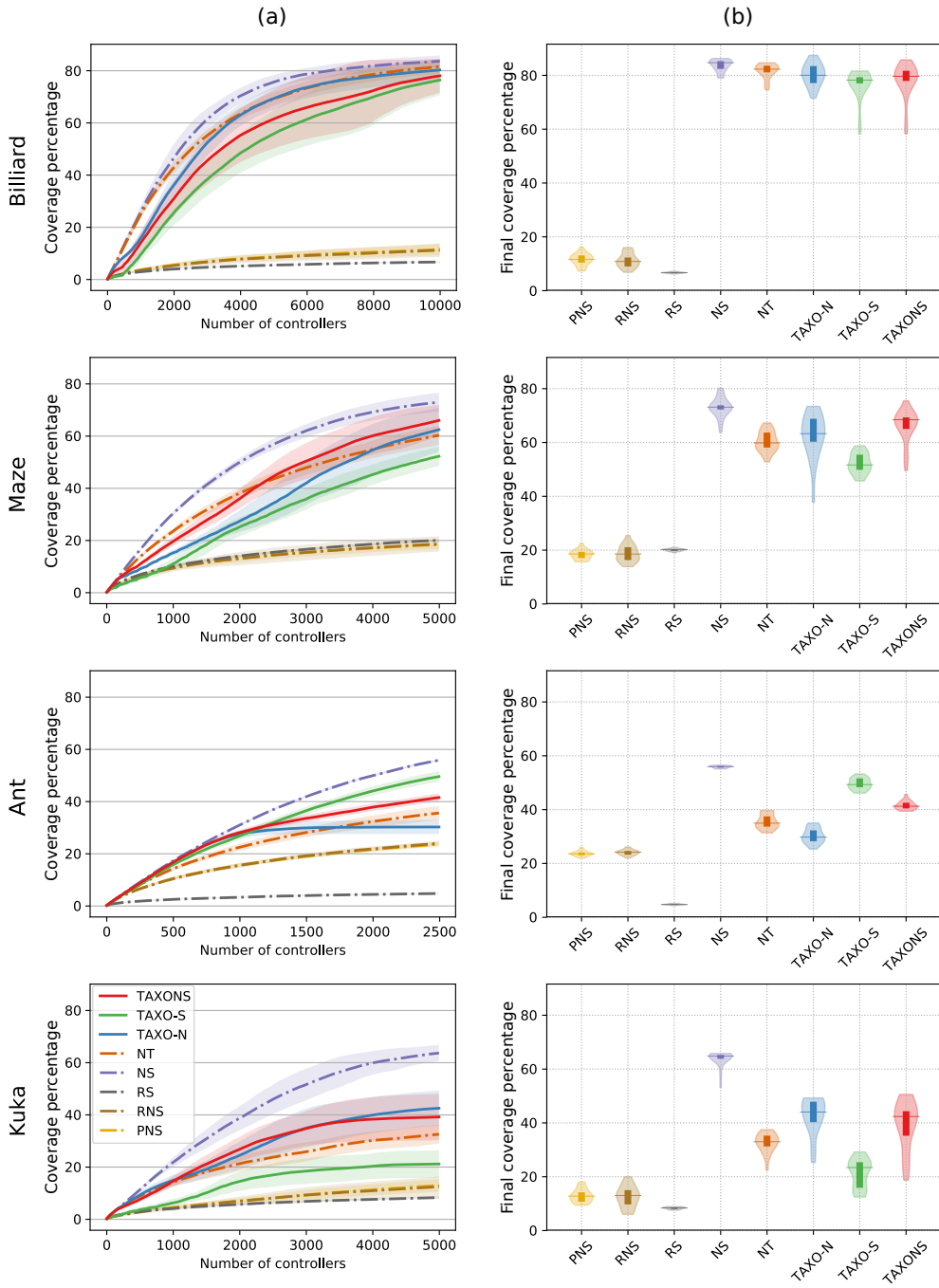
### 3.4.3 Results

The results displayed in Fig.3.6 show that TAXONS leads to a good coverage of the ground-truth  $(x, y)$  outcome space. Its performance is lower than the upper-bound performance of NS, which has direct access to the ground-truth outcome space, but significantly higher than the other baselines, which use as outcome descriptor the policy parameters (PNS), a random vector (RNS), or no outcome descriptor at all (RS). This shows that i) performing NS in a low-dimensional outcome space capturing informations about the final state of the system (through the last observation) is beneficial, and ii) that TAXONS can successfully build this space. Indeed when the generation and selection of policies is purely random (RS) the coverage is very low. Similarly, when low-dimensional outcome descriptors are randomly assigned to the policies (RNS) the coverage is only slightly better than purely random (Ant), or as bad (Billiard, Maze and Kuka). Finally, performing the NS directly in the high-dimensional policy parameters space  $\Theta$  (PNS), leads to a coverage that is similar to the RNS case. This suggests that, in these different experiments, performing the search in the high-dimensional policy parameter space is equivalent to assigning random descriptors to the policy, meaning that optimizing for diversity in the policy space does not help in exploring the behavior space. In contrast, the performance of TAXONS is significantly higher and more consistent in the four experiments (Billiard:  $p = 3.38 \times 10^{-8}$ , Maze:  $p = 3.38 \times 10^{-8}$ , Ant:  $p = 1.6 \times 10^{-7}$ , Kuka:  $p = 3.94 \times 10^{-8}$ ), showing that by learning the outcome space through an AE it is possible to capture relevant information about each system. The final performance of TAXONS (Billiard: 78.03, Maze: 66.02, Ant: 41.55, Kuka: 39.2) even approaches that of NS (Billiard: 83.66, Maze: 72.99, Ant: 55.88, Kuka: 63.9) although it remains inferior (Maze:  $p = 3.69 \times 10^{-5}$ , Ant:  $p = 7.65 \times 10^{-8}$ , Kuka:  $p = 3.39 \times 10^{-8}$ ),



**Figure 3.5:** (a) Coverage of the repertoire of policies found by TAXONS in the ground-truth  $(x,y)$  space. Highlighted in red are 3 policies for which the related final observation  $o_T$  are shown. (b) Sample policy from the repertoire generated by TAXONS. The trajectories followed by the policies are highlighted in red; for the Billiard, the trajectory of the point of the arm is also highlighted in blue. The final observation of the trajectory, with the respective reconstruction and descriptor generated by the AE, are also shown.





**Figure 3.6:** (a) Evolution of coverage metric over the number of policies in the repertoire. Note that the PSN and RNS curves are overlapping. (b) Final coverage measures for the tested methods.

especially for the Kuka environment, in which the skewed perspective requires the **AE** to encode also informations about the changing size of the cube rather than only its positions, rendering the task more complex, as the size covaries with the position. It must be highlighted that **NS** has direct access to the ground-truth  $(x, y)$  space, thus guaranteeing a very good performance.

Surprisingly, the **NT** baseline, in which the **AE** is never trained, performs similarly to **TAXONS** in the Billiard environment and only slightly worse in the other settings (Maze:  $p = 2.5 \times 10^{-4}$ , Ant:  $1.44 \times 10^{-7}$ , Kuka:  $p = 1.5 \times 10^{-3}$ ). This is probably due to the intrinsic power of convolutional neural networks in extracting relevant features in their inputs even when no training has been done, as shown in [155], and can be an interesting direction of investigation.

The performance of the two ablated versions (**TAXO-N** and **TAXO-S**) is similar to the one of **TAXONS**, as they lay between the **NS** upper-bound and the **PNS**, **RNS** and **RS** baselines. Nonetheless, their efficiency varies between experiments. **TAXO-S** performs similarly to **TAXONS** in the Billiard environment ( $p = 0.036$ ), worse in the Maze ( $p = 1.53 \times 10^{-6}$ ) and Kuka environments ( $p = 1.34 \times 10^{-6}$ ) and better in the Ant one ( $p = 7.69 \times 10^{-8}$ ). On the other hand, **TAXO-N** performs similarly to **TAXONS** in the Billiard, Kuka and Maze environments, while being significantly worse in the Ant one ( $p = 7.67 \times 10^{-8}$ ). After investigation, we hypothesize that the low performance of **TAXO-N** in the Ant environment is due to the specific dynamics of the **AE**. In the first phase of the training, the **AE** learns to reconstruct the large body of the agent while disregarding its legs. This leads to the outcome space temporarily capturing informations about the position of the agent in the  $(x, y)$  space, and thus allowing novelty search to better cover the ground-truth space. In a second phase, the **AE** learns how to reconstruct the legs. This leads the algorithm on exploring also different legs arrangements, rather than fully focusing on only covering the  $(x, y)$  space. **TAXO-S** performs significantly better in the same environment, as the impact of the body position on the reconstruction error is greater than the one of the legs. Thus maximizing the surprise also leads to maximizing the coverage.

From the results, it is possible to see that while the performances of the novelty and surprise are dependent of the dynamics of the environment, combining them renders **TAXONS** more robust than its two ablated versions. This allows performances almost as good as the ones of **NS**.

### 3.5 Conclusion

Studying the problem of sparse rewards, alongside the study of **QD** and divergent algorithms led to focusing the research effort towards the development of an algorithm that could learn and explore a search space with minimal supervision. **TAXONS** can generate a repertoire of diverse policies, without any external reward and with minimal prior knowledge about the system by exploring the space of reachable outcomes available. It does so by taking advantage of the exploration power of **NS** in a low-dimensional outcome space learned online by an **AE** trained directly on RGB images observations collected

during the search. Using these observations to autonomously learn the search space renders **TAXONS** easier to apply to different situations and tasks for which is not clear what the important features are.

The approach was tested on four different simulated environments and compared against different baselines. The results show that, by maximizing both novelty in the learned outcome space and surprise, derived from the **AE**'s reconstruction error, **TAXONS** finds a set of policies that covers the ground-truth outcome space, while being robust to different environments. At the same time, autonomously learning a **BS** means that the designer has no control on which information will be embedded in the representation. This can be a problem in more complex environments in which multiple aspects need to be represented in order to perform proper exploration. Given its nature, the algorithm will mainly focus on the most obvious aspects, while ignoring the smaller, but possible important ones. Moreover, the presence of *distractors* - elements not related to the ground-truth space to explore - can lead the algorithm to learn a poor representation. These and other limitations of similar methods will be discussed in depth in Chapter 7, together with the proposition of strategies to deal with them.

As said, the main objective of **TAXONS** is to explore as much as possible, but this is only a first step towards a more complete approach to sparse rewards problems. Once exploration is over, and rewards have been discovered, the algorithm should be able to take advantage of those rewards. At the same time, to address this problem, good exploration algorithms requiring minimal designer prior knowledge are fundamental. While **TAXONS** can efficiently explore and learn a low-dimensional representation of an unknown outcome space, it is still strongly limited by the restriction of working only on the last observation of the trajectory. This requirements limits the application of the algorithm only to environments whose last observation is informative enough with respect to the task. There are many ways in which this problem can be approached, some more naive than others. The next chapter will analyse one of these possible approaches, the *signature transform*, comparing its effectiveness to some simpler baselines.



---

**Chapter content**


---

<b>4.1</b>	<b>Introduction</b>	<b>48</b>
<b>4.2</b>	<b>The signature transform</b>	<b>49</b>
4.2.1	Signature of a discrete path	53
<b>4.3</b>	<b>Signed Behavior Descriptor</b>	<b>53</b>
<b>4.4</b>	<b>Experiments</b>	<b>56</b>
<b>4.5</b>	<b>Results</b>	<b>57</b>
4.5.1	Exploration	57
4.5.2	Rewards	59
<b>4.6</b>	<b>Conclusion</b>	<b>60</b>

---

## 4.1 Introduction

Chapter 3 discussed a big limitation that QD algorithms, and NS in particular have: the BS, that is the space in which the behavior of the policies is represented and the diversity is measured, needs to be hand-designed. To address this problem, TAXONS was introduced. This is an algorithm that can learn, online and in an unsupervised way, a BS representative of the outcome of each policy, allowing to reduce the amount of prior information and the design effort needed to solve a task. Notwithstanding the promising results of TAXONS, the method still relies on a strong assumption: the last observation of a trajectory should contain enough information to describe the behavior of a policy. In situations in which the most informational moment of the policy behavior happens at an unspecified time during the evaluation, the exploration process can be less efficient, preventing the discovery of good solutions. An example of this can be a robot trying to launch a ball against a wall, from which it can bounce back. The final position reached by the ball is not informative of the position in which it hit the wall. To properly explore, the method should take into account the moment in which the ball hits the wall, but this is difficult to know. This makes the maximization of the diversity of these positions complicated. For this reason, in these situations the algorithm should be able to push for diversity along the whole trajectory of traversed states, not only on the last one.

Pushing for diversity over the space of trajectories requires a way to calculate a distance between different trajectories. There are two strategies that can be followed. The first one consists in calculating the distance between the whole trajectories, either state-by-state, or through the *dynamic time warping* method [156]. The calculation of the distance state-by-state is not always feasible because the trajectories can be of different lengths. At the same time the dynamic time warping can be extremely slow when comparing distances between thousands of trajectories, as it happens for NS based methods. The other possible approach consists in first obtaining a compressed representation, or encoding, of the trajectories and then calculating the distance on said compressed representations. This has the advantage of reducing the dimensionality of the space in which the distance is calculated, preventing many of the issues related to distance metrics in high-dimensional spaces [143]. However, these approaches add the additional complexity of calculating the compressed representation of the trajectory. There are multiple way to do so, ranging from a naive sub-sampling of the trajectory, to more sophisticated but also more computationally intensive machine learning approaches like *Long Short-Term Memory (LSTM)* encoders [157].

In this chapter, another method to perform the trajectory encoding will be analyzed: the *signature transform*. This transform, introduced by Chen [50, 51, 52], is a mathematical object defined in the scope of rough field theory. It allows to describe a stream of data as a single vector, the signature, obtained through an infinite series of integrals. In practice the signature is calculated up to a certain order of the infinite series of integrals, proving a quick and easy way to compress a sequence of data to any desired precision. This, and the other useful properties that the signature transform enjoys, have attracted the attention of the machine learning community over this method with the goal of encoding sequences of data in a fast and easy way [53, 54].

The signature transform can be used to describe the policies behaviors. This is done by compressing the whole trajectory of states traversed by the system during the policy evaluations into a smaller behavior descriptor vector. The goal is to obtain more diverse solutions over the whole space of trajectories, rather than only over the final outcome. Moreover, this approach will reduce even more the amount of prior information needed at design time, allowing the application of QD methods to new and more complex domains.

The chapter is structured as follows. Sec. 4.2 will present the signature transform, while Sec. 4.3 will describe how it can be applied to NS. The testing environment will be presented in Sec. 4.4 and the results discussed in Sec. 4.5. The chapter will conclude with Sec. 4.6 in which the lessons learned during this analysis will be discussed.

## 4.2 The signature transform

This section will describe in a formal way what the signature transform of a sequence of data is and what its properties are.

Let's start by noting that a sequence of data  $\mathbf{x} = (x_0, \dots, x_T)$  of length  $T$ ,

where  $x_i \in \mathbb{R}^d$ , can be described by a continuous function:

$$f : [0, 1] \rightarrow \mathbb{R}^d, \quad \text{such that } f\left(\frac{i}{T}\right) = x_i \in \mathbb{R}^d. \quad (4.1)$$

Moreover, the function  $f$  can also be represented as the list of its components along each dimension:

$$f(t_i) = \{f^1(t_i), \dots, f^d(t_i)\}, \quad (4.2)$$

with  $t_i = i/T$ . Thanks to this function, the study of a sequence of data  $x$  can be viewed as the geometrical study of the path of  $f(\cdot)$  in the space  $\mathbb{R}^d$ . For example the changing temperature of a room during the day can be seen as a path in  $\mathbb{R}$ ; the motion of an ant going back to its nest as a path in  $\mathbb{R}^2$ ; the changes within  $d$  indices of financial markets as a path in  $\mathbb{R}^d$ ; etc. [53].

These functions are called *paths* in the field of rough field theory. The goal of the signature transform is to study the geometrical characteristic of these paths. This is done by the iterated application of integrals on the function, in order to obtain a vector encoding a collection of geometrical statistics of  $f(\cdot)$ . This vector is called *signature*. The method allows to extract an accurate summary of the path and to obtain arbitrarily good linear approximations of continuous functions of paths [158].

The main assumption needed in order to calculate the signature of these functions is that these paths are of *bounded variation*, i.e. the first derivative of the function exists almost everywhere. More formally, a function  $f$  is of bounded variation if its total variation is finite. The total variation is expressed as:

$$\|f\|_{1-var} = \sup_D \sum_{t_i \in D} |f(t_i) - f(t_{i-1})|, \quad (4.3)$$

where the supremum is calculated over all the finite partitions:

$$D = \{(t_0, \dots, t_T) | T \geq 1, 0 = t_0 < t_1 < \dots < t_{T-1} < t_T = 1\}, \quad (4.4)$$

with  $t_i = i/T$ .

To define the signature, let's start by considering the simplest case of a path for which  $d = 1$ . In this case the path would be  $f(t_i) = \{f^1(t_i)\}$ , for which its integral would be:

$$\mathcal{S}(f)_{0,T}^1 = \int_{0 \leq t \leq T} df^1(t) = f^1(T) - f^1(0). \quad (4.5)$$

This integral is what it is called the *1-fold iterated integral* of the path and is a real valued integral representing the increment, or the displacement, of this one-dimensional path over the whole time interval [159].

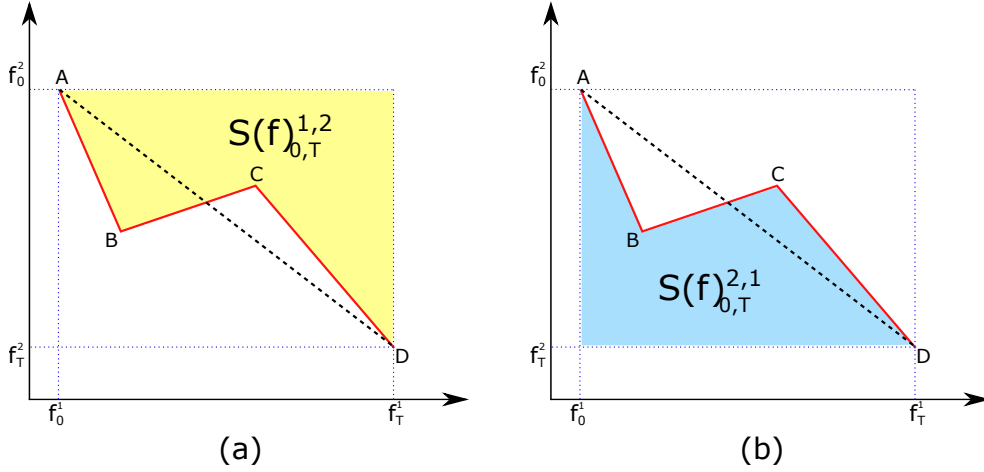
The *2-fold iterated integral* of the path can be obtained by reapplying the same integral operator:

$$\mathcal{S}(f)_{0,T}^{1,1} = \int_{0 < t_2 \leq T} \mathcal{S}(f)_{0,t_2}^1 df^1(t_2) = \frac{1}{2} \left( f^1(T) - f^1(0) \right)^2, \quad (4.6)$$

which represents the square of the increment of the path over the time interval. The recursive application of this operator leads to the definition of the  $k$ -fold iterated integral, which is proportional to the increment of the path to the power of  $K$ :

$$\begin{aligned} \mathcal{S}(f)_{0,T}^{I_K} &= \int_{0 < t_K \leq T} \cdots \int_{0 < t_2 \leq t_3} \int_{0 < t_1 \leq t_2} df^1(t_1) df^1(t_2) \cdots df^1(t_K) \\ &= \frac{1}{K!} \left( f^1(T) - f^1(0) \right)^K, \end{aligned} \quad (4.7)$$

where  $I_K = (1, 1, \dots, 1)$  is the set of indexes of size  $K$ .



**Figure 4.1:** Geometric intuition of the 2-fold iterated integral of a 2D path as shown by Yang et al. [159]. The path in red moves from point A to D during a time interval of  $[0, T]$ . The chord connecting the endpoints is shown as a dashed line. Figure (a) and (b) show the last two elements of the 2-fold integral of the path.

In the case of a path of  $d = 2$ , the path would be  $f(t_i) = \{f^1(t_i), f^2(t_i)\}$ . This leads to a 1-fold iterated integral containing two elements, the same as in Eq. 4.5 but for each single dimension:

$$\mathcal{S}(f)_{0,T}^1 = \int_{0 \leq t \leq T} df^1(t) = f^1(T) - f^1(0), \quad (4.8)$$

$$\mathcal{S}(f)_{0,T}^2 = \int_{0 \leq t \leq T} df^2(t) = f^2(T) - f^2(0). \quad (4.9)$$

At the same time, the 2-fold iterated integral will contain  $d^2 = 4$  elements,



due to the cross-terms between the dimensions:

$$\mathcal{S}(f)_{0,T}^{1,1} = \int_{0 < t_2 \leq T} \mathcal{S}(f)_{0,t_2}^1 df^1(t_2) = \frac{1}{2} \left( f^1(T) - f^1(0) \right)^2, \quad (4.10)$$

$$\mathcal{S}(f)_{0,T}^{2,2} = \int_{0 < t_2 \leq T} \mathcal{S}(f)_{0,t_2}^2 df^2(t_2) = \frac{1}{2} \left( f^2(T) - f^2(0) \right)^2, \quad (4.11)$$

$$\mathcal{S}(f)_{0,T}^{1,2} = \int_{0 < t_2 \leq T} \int_{0 < t_1 \leq t_2} df^1(t_1) df^2(t_2), \quad (4.12)$$

$$\mathcal{S}(f)_{0,T}^{2,1} = \int_{0 < t_2 \leq T} \int_{0 < t_1 \leq t_2} df^2(t_1) df^1(t_2). \quad (4.13)$$

The first 2 correspond to the square of the displacement caused by the path for each dimension, as in Eq. 4.6 for the 1 dimensional path. At the same time,  $\mathcal{S}(f)_{0,T}^{1,2}$  and  $\mathcal{S}(f)_{0,T}^{2,1}$ , shown respectively in Fig. 4.1.(a) and Fig. 4.1.(b), encode information about the area covered by the path.

This example can be generalized to the  $k$ -fold integral of a path in  $\mathbb{R}^d$ . Such an integral will have  $d^K$  components that can be expressed as:

$$\mathcal{S}(f)_{0,T}^{I_K} = \int_{0 < t_K \leq T} \cdots \int_{0 < t_2 \leq t_3} \int_{0 < t_1 \leq t_2} df^{i_1}(t_1) df^{i_2}(t_2) \cdots df^{i_K}(t_K), \quad (4.14)$$

where  $I_K = (i_1, \dots, i_K)$  is the set of indexes of size  $K$ , with  $i_j \in \{1, \dots, d\}$ .

Finally, the signature of a path  $f(\cdot)$  can be defined. This will be the collection of all the iterated integrals over the path and can be expressed as:

$$\begin{aligned} \text{Sig}(f)_{0,T} = & \left( 1, \mathcal{S}(f)_{0,T}^1, \dots, \mathcal{S}(f)_{0,T}^d, \right. \\ & \mathcal{S}(f)_{0,T}^{1,1}, \dots, \mathcal{S}(f)_{0,T}^{1,d}, \mathcal{S}(f)_{0,T}^{2,d}, \dots, \mathcal{S}(f)_{0,T}^{d,d}, \\ & \left. \dots, \mathcal{S}(f)_{0,T}^{1,1,\dots,1}, \dots, \mathcal{S}(f)_{0,T}^{d,d,\dots,d}, \dots \right), \end{aligned} \quad (4.15)$$

where the first element is set to 1 as a convention. Being calculated over all possible combination of indices of finite length, the signature is a vector of infinite size. In practice, it has to be truncated up to a certain level, or *order*,  $K$ :

$$\text{Sig}^K(f)_{0,T} = \left( 1, \mathcal{S}(f)_{0,T}^1, \dots, \mathcal{S}(f)_{0,T}^d, \dots, \mathcal{S}(f)_{0,T}^{i_1, i_2, \dots, i_K}, \dots, \mathcal{S}(f)_{0,T}^{d, d, \dots, d} \right). \quad (4.16)$$

In this case, the dimensionality of  $\text{Sig}^K(f)_{0,T}$  is:  $(d^{K+1} - d)(d - 1)^{-1}$ . From this two main observations can be made:

1. The size of the signature is fixed with respect to the length of the path. This property is extremely useful in situations in which paths of different lengths might have to be compared one against the other, as is the case of [NS](#);
2. The size of the signature increases exponentially with  $K$  and polynomially with  $d$ . This means that the right balance between the size of the signature and the amount of information preserved when truncating it needs to be found.

### 4.2.1 Signature of a discrete path

As discussed, the path signature is defined for continuous paths with bounded variation. This means that, in case of a sequence of sampled points, the path function defined in Eq. (4.1) needs to perform interpolation between successive points of the sequence. This operation is called *embedding* and it can be done in multiple ways [54]. This work will consider two among them:

- **Linear:** consisting of connecting each consecutive points by a straight line. This is one of the most used interpolation in the literature thanks to its simplicity [160, 54, 159, 53];
- **Time Path:** building over the linear path, it adds another monotone coordinate encoding the time [159, 54]. Having a monotone coordinate allows to have an unique signature for each path. This is because the path will contain information also about its speed rather than only its geometry. However, this comes at the cost of having an additional coordinate to the path, leading to higher dimensional signatures.

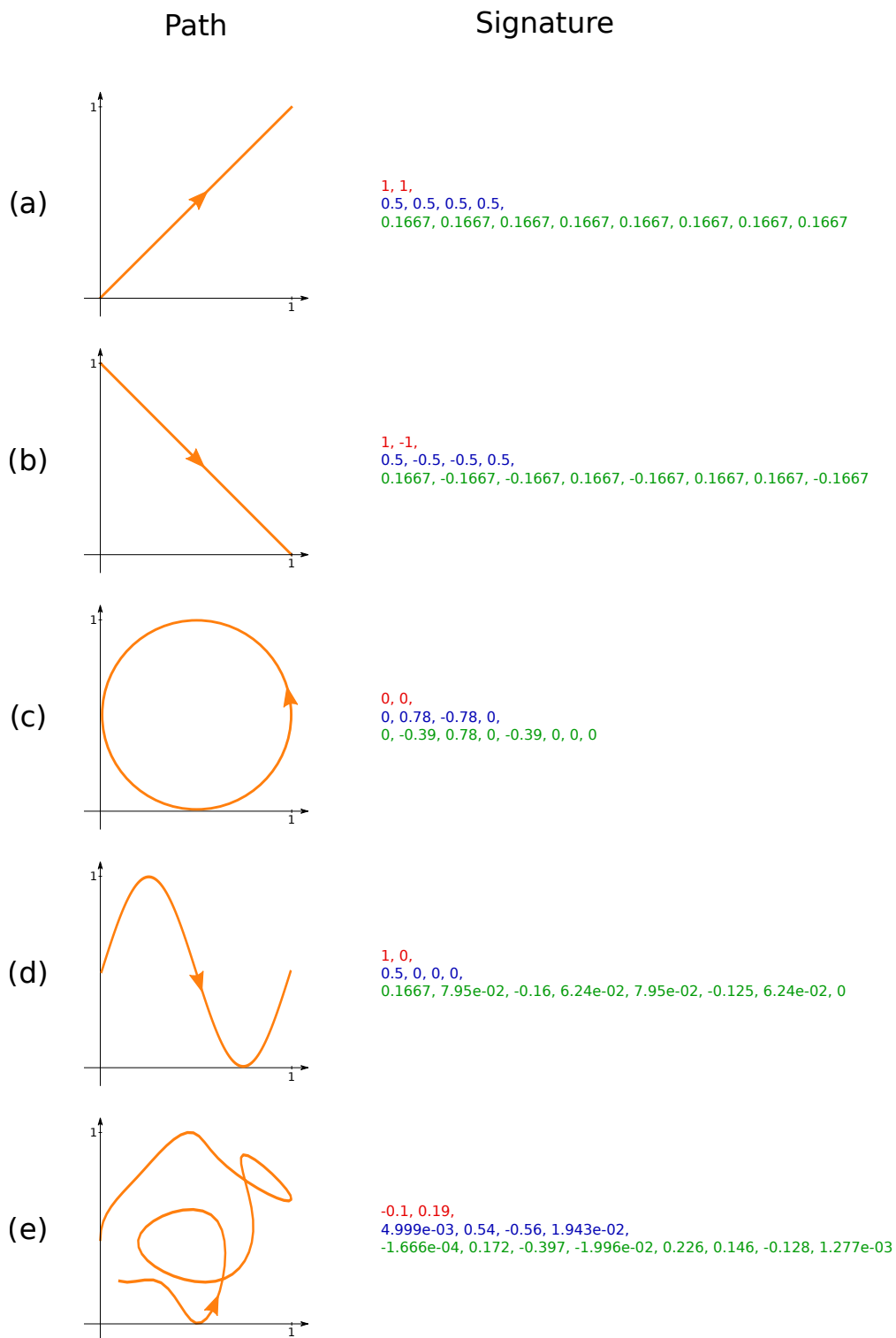
An example of the signature of order 3 for different oriented paths with linear embedding is shown in Fig. 4.2. The distances calculated between the signatures of these different paths are shown in Tab. 4.1. It is possible to see how, in the signature transform space, the two straight lines Fig. 4.2.(a) and 4.2.(b) are the farthest ones. At the same time, they are both very close to the sinusoidal in Fig. 4.2.(d), thanks to this having a trend piecewise similar to either two of the straight lines. The other closest curves in the signature space are the circle (Fig. 4.2.(c)) and the more complex path (Fig. 4.2.(e)). This is likely due to the complex path having starting and finishing positions very close one to the other. From this it is possible to observe that the signature transform, and the distance in the corresponding space, can encode multiple - and not immediately evident - aspects of a path.

## 4.3 Signed Behavior Descriptor


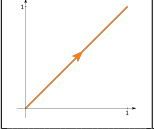
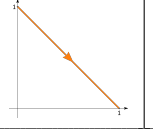
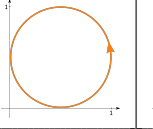
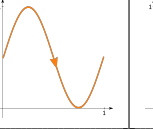

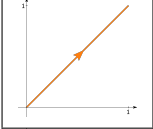
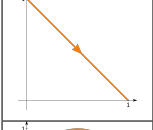
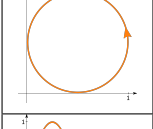

Extracting the behaviour descriptor from the trajectory of observations requires the knowledge of the time-step at which the interesting behaviour has happened. This can be limiting in situations in which the time-step is not always the same or when multiple events can happen at different time-steps.

Using the whole trajectory of observations as behaviour descriptor can help in overcoming this problem. At the same time, this requires to encode this trajectory in a lower dimensional space to prevent the degeneration of the distance metrics that happens in high-dimensional spaces [143]. The signature transform provides exactly this while being fast to compute and having strong mathematical guarantees.

This means that the signature transform can be used to extract the behavior descriptor  $b_i$  used by NS to evaluate the novelty of a policy  $\theta_i \in \Theta$ . In section 2.2.2, it was shown how a policy  $\theta_i$  traverses a trajectory of states  $\tau = [s_0, \dots, s_T]$  when evaluated. The states are observed through some sensors



**Figure 4.2:** Signature of degree 3 of different oriented paths. The paths are shown in orange. Each line of the signature represents the elements of a different order: in red the elements of the first order, in blue of the second and in green of the third.

					
	0	2.538	2.313	1.417	1.991
		0	2.313	1.416	2.165
			0	1.96	1.507
				0	1.526

**Table 4.1:** Distances between the signatures of the curves shown in Fig. 4.2.

to generate a corresponding trajectory of observations  $\tau_{\mathcal{O}} = [o_0, \dots, o_T]$ . NS then uses an observer function  $O_B : \mathcal{O}^T \rightarrow \mathcal{B}$  to extract the behavior descriptor from the trajectory of observations. Usually the observer function only works on some selected observations from the whole trajectory, forcing the designer to know at which time-step the most interesting observations can be found. Using the truncated signature transform  $\text{Sig}^K(\cdot)_{0,T}$  as observer function to encode the whole trajectory into  $b_i$  can remove this limitation. This means that Eq. (2.5) can be rewritten as:

$$\phi(\theta_i) = \text{Sig}^K(\tau_{\mathcal{O}}^i)_{0,T} = b_i, \quad (4.17)$$

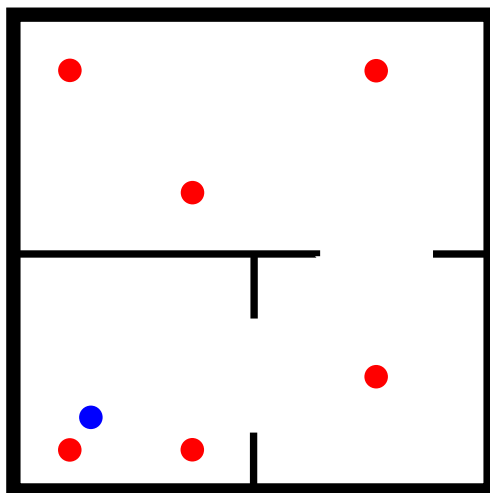
where  $\tau_{\mathcal{O}}^i$  is the trajectory of observations generated by the evaluation of the policy  $\theta_i$ .

To be able to use the signature in this way, the constraint defined at the beginning of Sec. 4.2 has to be respected. This requires the function defined in Eq.(4.1), representing the trajectory of observations, to be of bounded variation. In practice, this constraint is not too restraining: as long as  $o_t \in \mathbb{R}^d$ , the simple linear embeddings defined in Sec. 4.2.1 will always limit  $f(\cdot)$  according to:

$$-\infty < \min(\tau_{\mathcal{O}}) \leq f(\cdot) \leq \max(\tau_{\mathcal{O}}) < \infty. \quad (4.18)$$

Note that the most external inequalities are always true thanks to the closeness of  $\mathbb{R}^d$  with respect to addition and multiplication, meaning that neither  $-\infty$  nor  $\infty$  are included in the set of the real numbers.

This method will be tested in the next sections, to study if a signature-based behavior descriptor can help the exploration process of NS in discovering



**Figure 4.3:** *The CollectBall environment. In blue is shown the 2 wheeled robot, while in red are the 6 balls to be collected. The collection point corresponds to the same position of where the robot starts the episodes.*

a more diverse set of solutions with respect to classical descriptors. Note that, while the final goal is to apply the discussed ideas to **TAXONS**, the methods in this Chapter will be tested on the ground-truth low-dimensional observations of the environment. This allows to better focus on the properties of the methods themselves, without any possible interference from the dimensionality reduction component of **TAXONS**.

## 4.4 Experiments

The *signature based behavior descriptor* for **NS** is tested in this section. The question this study wants to answer is:

*Is the signature encoding of the whole trajectory of traversed states beneficial to the exploration performed by **NS**?*

In light of this, the algorithm is tested in the CollectBall environment [113], shown in Fig. 4.3. This consists of a maze in which a 2-wheeled robot has the goal of collecting 6 red balls. The robot, in blue in the figure, is controlled by a 2 layers **NN**, with each layer of size 5. The controller receives a 10 dimensional state vector as input and outputs a 3 dimensional control vector. The input state vector consists of the reading of the sensors with which the robot is equipped: 3 distance sensors, 2 bumpers, 2 ball detection sensors, 2 goal detection sensors and a "collected ball" sensor. At the same time, the control vector returned by the **NN** consists of the speed of the 2 wheels and a "collect ball" signal. If the robot is close to a ball and the value of this signal is over 0.5 the ball is collected; if the robot is carrying a ball and the value of this signal is lower than 0.5, the ball is released. The robot can carry a

single ball at a time. A ball is considered collected if it is released close to the basket located at the initial position of the robot. On the contrary, if a ball is released away from the basket, it appears in the maze at the position it was released. Each policy is run in the environment for 2000 timesteps. The final reward given to the agent is equal to the number of collected balls at the end of an episode.

The following variants and baselines are compared in the experiments:

- **NS**: vanilla **NS**, whose behavior descriptor is calculated as the final  $(x, y)$  position of the robot in the map;
- **NS\_multi**: **NS** whose behavior descriptor is calculated by concatenating 5 vectors, sampled at regular intervals during each trajectory, containing the  $(x, y)$  position of the robots. The descriptor has size 10;
- **SIGN**: **NS** in which the behavior descriptor is the signature of order  $K = 5$  of the trajectory of the robot. This baseline uses the *linear* embedding: each datapoint along the trajectory consists of the  $(x, y)$  position of the robot at the corresponding time-instant. This leads to a descriptor of size 62;
- **TIME-SIGN**: **NS** in which the behavior descriptor is the signature of order  $K = 5$  of the trajectory of the robot. This baseline uses the *time path* embedding: each datapoint of the trajectory is composed by concatenating the  $(x, y)$  position of the robot and the time coordinate  $t \in [0, 1]$ . This gives a descriptor of size 363.

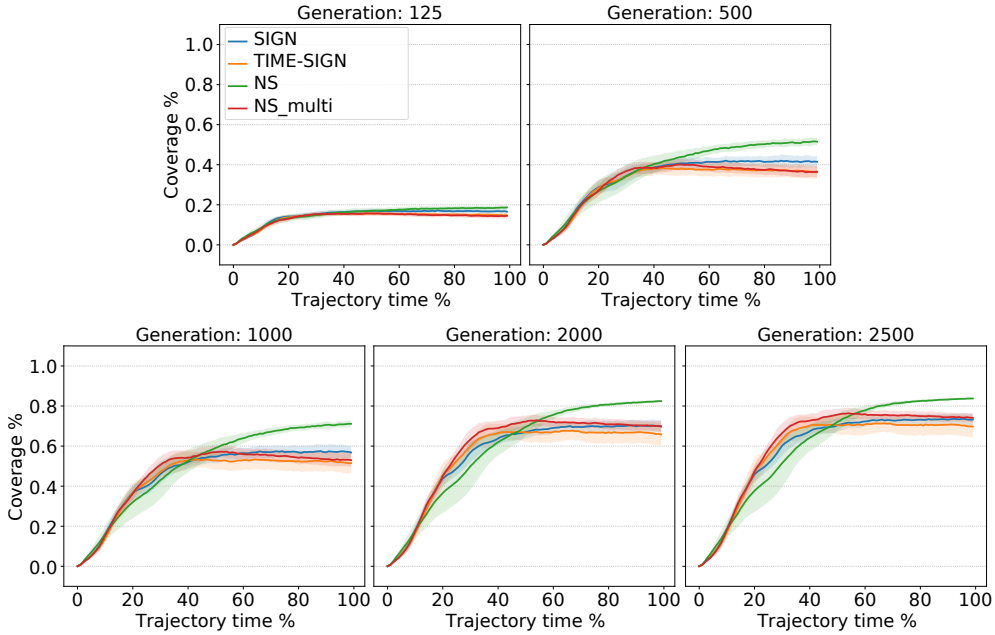
Finally, the statistical results are computed over 15 runs for each experiment.

## 4.5 Results

This section discusses the results obtained during the experiments.

### 4.5.1 Exploration

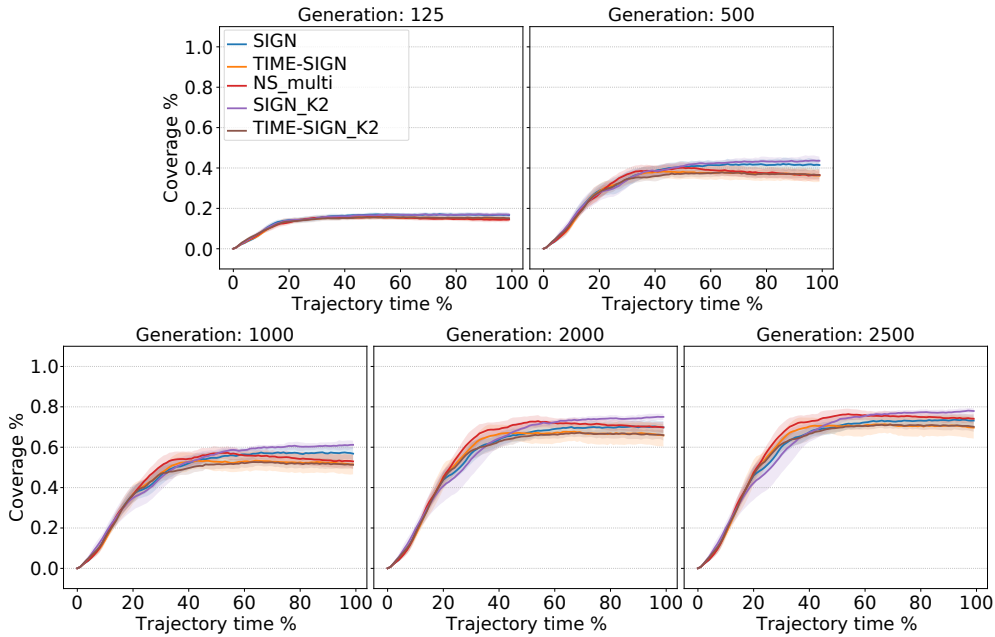
The first thing to verify is the ability of the signature transform to drive exploration for **NS**. To do so, this section studies how well the population can cover the **BS** at any given moment while performing the trajectory. This means that the trajectories are discretized in 100 equidistant timesteps and the coverage is calculated at each timestep. The coverage itself is measured by calculating the percentage of occupied cells of a  $50 \times 50$  grid discretizing the  $(x, y)$  space in which the robot moves [43, 48]. The results for different generations during the experiments are shown in Fig. 4.4. For the first generations all the methods perform similarly, as can be see for generation 125 in the Figure. With the passing of the generations a clear trend starts to emerge, with the vanilla **NS** variant reaching the highest coverage at the end of the trajectories ( $p = 3.93 \times 10^{-11}$ ). This is not surprising, given that this version optimizes for the higher diversity at the end of the trajectories themselves. The other



**Figure 4.4:** Average coverage with respect to the time of the trajectory for 5 different generations. The shaded areas represent one standard deviation.

methods optimizing for diversity along the whole trajectories reach higher coverages earlier in the trajectory than **NS**. **NS\_multi** can cover almost 80% of the whole space just after only 30% of the total timesteps. **NS** reaches the same values only after more than 50% of the trajectories have been executed. The signature based variants can also explore in a similar fashion to **NS\_multi**, reaching high coverage values early on in the trajectories. This is also expected: contrary to vanilla **NS**, these variants are designed to diverge along the whole time dimension. At the same time, the signature based variants perform worse or similarly than the simpler **NS\_multi**. While early on in the exploration, at generation 500, **SIGN** reaches higher coverage than both **NS\_multi** and **TIME-SIGN** ( $p = 1.81 \cdot 10^{-4}$ ), by the end of the run, at generation 2500, both **SIGN** and **TIME-SIGN** perform worse or similar than **NS\_multi** ( $p = 0.389$  for **SIGN** and  $p = 9.58 \cdot 10^{-3}$  for **TIME-SIGN**). This phenomenon happens along the whole trajectory.

A possible explanation of this effect is the higher dimensionality of the descriptor generated by the signature methods. Aggarwal et al. showed how the euclidean distance, used to calculate the novelty of the policies' behaviors in Eq. (2.6), loses meaning in high-dimensional spaces. To test this hypothesis experiments with variants of both **SIGN** and **TIME-SIGN** with a lower order of the transform:  $K = 2$  rather than  $K = 5$  have been performed. While this lowers the amount of information about the structure of the trajectory included in the descriptor, it also reduces the dimension of the descriptor. The two variants are called **SIGN\_K2** and **TIME-SIGN\_K2**, with a descriptor dimensionality of 6 and 12, respectively. The coverage results are shown in Fig. 4.5. It is possible to see that while the **TIME-SIGN** variants perform in



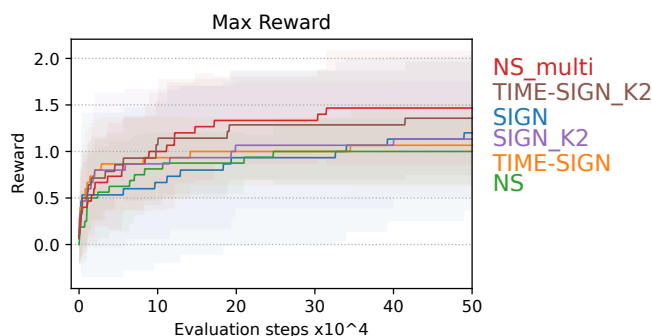
**Figure 4.5:** Average coverage with respect to the time of the trajectory for 5 different generations. The shaded areas represent one standard deviation.

a very similar fashion, the `SIGN_K2` tends to reach higher coverages towards the end of the trajectory ( $p = 1.74 \times 10^{-05}$ ). This is due to the fact that the signature of this order takes into account only the displacement caused by the trajectory and the area covered, thus not being able to encode all the variations in the path that can happen between different policies. A possible reason for this is that the loss of information due to a lower order of the signature balances out any improvements obtained by the reduction in dimensionality of the descriptor. At the same time, the `NS_multi` still reaches the highest coverage from early on in the trajectory, notwithstanding having similar final coverage with `SIGN`. This shows how such a simple method can outperform the more complex signature transform when using it in combination with `NS` to drive exploration. This can be due to the signature fostering exploration in a space not relevant to the task, thus different than the one in which the coverage is calculated and `NS_multi` operates.

#### 4.5.2 Rewards

This section studies the reward obtained by the tested variants in the environment. Note that none of the methods used in this and in the previous chapter have an explicit way of optimizing the reward. This means that any obtained reward is only a by-product of the exploration process. However, an exploration process capable of obtaining higher rewards is extremely useful in sparse reward settings. This is because the method can easily be used in conjunction with a reward exploitation approach to optimize the rewarding solutions, as discussed in Chapter 5.





**Figure 4.6:** Average maximum reward. The shaded areas represent one standard deviation

In this setting, the reward corresponds to the amount of balls collected by the robot. The idea behind this is that a diverse enough set of trajectories should be able to collect more balls than policies diversified only with respect to the final position of the robot. The average over 15 runs of the maximum reward collected with respect to the policies evaluations of each variant is shown in Fig. 4.6. It is possible to see how **NS** on average tends to collect less than 1 ball, while the variants optimizing for diversity over the whole trajectories reach higher rewards. The best performing one is **NS\_multi**, proving a more effective method than the signature based ones in performing exploration geared towards sparse rewards systems.

## 4.6 Conclusion

This chapter analyzed a way to reduce the amount of prior information needed by the **TAXONS** algorithm. Removing the reliance of the method - and **NS** based methods in general - on a specific observation to calculate the behavior descriptor of a policy can greatly extend the generalizability of these algorithms. In order to do so, the signature transform was tested as a way to encode information about the whole trajectory of traversed states into the behavior descriptors used to calculate the policies novelties. This novelty would be calculated then not only on a single state, but with respect to the whole trajectory. The idea has been inspired by the success of the application of the signature transform in other domains of machine learning to encode a stream of data into a single vector [54, 53]. Before applying the transform, the stream of data can be *embedded* in multiple ways. Two of the most commonly used embeddings were considered: the linear and the time path. This was compared against two other approaches of calculating the policy behavior descriptor. The first one uses only the final observation. The other variant concatenates multiple states sampled at regular intervals along the trajectory. Notwithstanding its simplicity, this last variant has shown to perform better than all the signatures based variants tested. A possible reason for this could be the high dimensionality of the extracted features by the signature transform. This was tested by running experiments with a lower order signature.

The obtained results moved in the direction of confirming our hypothesis for the variant with the linear encoding, but they also showed that the amount of information discarded by the lower order tends to be too high to properly diversify the behaviors along the whole trajectories. Finally, a test on which of the variants could push for an exploration able to discover as much reward as possible was conducted. This is a fundamental aspect in sparse rewards settings. Once again the simpler method of stacking multiple observations proved to be the best performing one. At the same time, these methods have only been tested on a single setup. More experiments should be performed on a bigger variety of environments in order to confirm these results.

Nonetheless, in Chapter 6 the simpler `NS_multi` strategy was used in order to remove the limitation due to `TAXONS` working only on the last observation of the trajectory. Another limiting aspect of the signature approach together with `TAXONS` is that in order to keep the dimension of the signature vector acceptable, the high-dimensional observations need to be encoded first by the `AE`. This would greatly increase the computational cost of the method, having to use the encoder on all the observations of the trajectory rather than just the few sampled ones for the `NS_multi` approach.

At the same time, this does not disqualify the use of the signature transform with divergent search algorithms. More research should be done in this regard, also in light of the usefulness that this method has shown in different domains of machine learning research for the embedding of streams of data.

In the last two chapters, methods for reducing the amount of task-specific prior information needed at design time for `NS` have been proposed. The next chapter will focus on how to take advantage of the rewards discovered during the search. This will be done by introducing a method augmenting `NS` with the capability of focusing on these rewards to optimize the policies with respect to them.



**Chapter content**

---

<b>5.1</b>	<b>Introduction</b> . . . . .	<b>63</b>
<b>5.2</b>	<b>Emitters</b> . . . . .	<b>64</b>
<b>5.3</b>	<b>Method</b> . . . . .	<b>65</b>
	5.3.1 Exploration phase . . . . .	67
	5.3.2 Exploitation phase . . . . .	68
<b>5.4</b>	<b>Experiments</b> . . . . .	<b>72</b>
<b>5.5</b>	<b>Results</b> . . . . .	<b>74</b>
	5.5.1 Budgeting . . . . .	74
	5.5.2 Exploration . . . . .	76
	5.5.3 Exploitation . . . . .	77
	5.5.4 Final archive distribution . . . . .	77
<b>5.6</b>	<b>Conclusion</b> . . . . .	<b>80</b>

---

This chapter is adapted from the following publication:

*Paolo, G., Coninx, A., Doncieux, S., & Laflaquière, A. Sparse Reward Exploration via Novelty Search and Emitters.* In *The Genetic and Evolutionary Computation Conference 2021 (GECCO 2021)*.

---

**5.1 Introduction**

When **NS** was discussed in chapter 2, it was highlighted how well the algorithm can explore and cover the search area by ignoring any potential reward and tending towards a uniform exploration of the search space [125]. At the same time, its strength is also its limitation: considering all the non-rewarding areas of the search space as valuable as the rewarding ones prevents the algorithm from finding the best possible solutions for solving the task. Augmenting **NS** with the ability to shift its focus from pure exploration to reward exploitation could help address this issue. One possible way of doing so is by using multi-objective optimization methods like NSGA-II [115] that can focus both on the diversity and on the reward at the same time. However, as it will be shown

in the following, merging exploration and exploitation through a Pareto front can degrade the exploring power of the algorithm. A different approach is taken by QD algorithms, a family of methods that build a set of both diverse and high-quality solutions [41].

In this chapter, the SparsE Reward Exploration via Novelty and Emitters (SERENE) method [161] is introduced. This is a QD algorithm explicitly designed to address sparse reward problems. SERENE augments NS with *emitters* [55] to perform rewards maximization while keeping its exploration ability, thanks to a clear separation between exploration and exploitation. Introduced as a way to improve the efficiency of ME [43] in the CMA-ME method [55], *emitters* are instances of reward-based algorithms scheduled to perform a local optimization in the search space. In the original formulation, ME acts as a scheduler by initializing emitters in different areas of the search space. The emitters then perform both local exploration and exploitation of the reward, leading to degraded performances in settings with very sparse rewards, where not all policies can obtain a reward. Conversely, SERENE decouples exploration from exploitation to better deal with these situations. The former is performed through NS, completely ignoring the reward. Once a reward area is found, SERENE spawns emitters focusing solely on its maximization. This allows minimal interference between the two processes of exploration and exploitation while letting our algorithm shift its focus between the two processes at any moment. Persisting in exploring even after some reward areas have been found is essential, since other reward areas could be present in the search space.

In the following, emitters and how they work will be presented in detail in Section 5.2. SERENE itself will be introduced in Section 5.3, tested in Section 5.4, and the results discussed in Section 5.5. The Chapter will conclude with Section 5.6, pointing at possible extensions and improvements.

## 5.2 Emitters

An emitter [55, 56] is an instance of an reward optimization algorithm. Its objective is to rapidly examine a small area of the search space while optimizing on the reward. In the work from Fontaine et al. [55] and Cully [56] the CMA-ME algorithm combines CMA-ES-based emitters [162] with ME [43], by using the latter as a scheduler for the emitters evaluation. It works by initializing a population of policies  $\theta$  by sampling their parameters from a distribution  $\mathcal{N}(\mu, \Sigma)$  and adding them to the ME archive. The algorithm then samples one of these policies and uses it to initialize the population of the emitter  $\mathcal{E}_i$ . At this point,  $\mathcal{E}_i$  is evaluated until a termination criterion is met; e.g. a lack of increase of the reward found. Moreover, the policies found during the evaluation of the emitter are added to the ME archive according to ME addition strategy. After the termination of  $\mathcal{E}_i$ , a new emitter is initialized by sampling another policy from the archive. This is repeated until the whole evaluation budget is depleted.

Different types of algorithms can be used as emitters, changing how the

search is performed and how the policies are selected. This shows the flexibility of the approach. At the same time, existing methods perform exploration through reward-following emitters [55, 56]. This reduces performances in situations where the reward is very sparse and many of the policies do not get any reward. Decoupling the exploitation of the reward from the exploration allows to more efficiently deal with sparse rewards settings [100].

### 5.3 Method

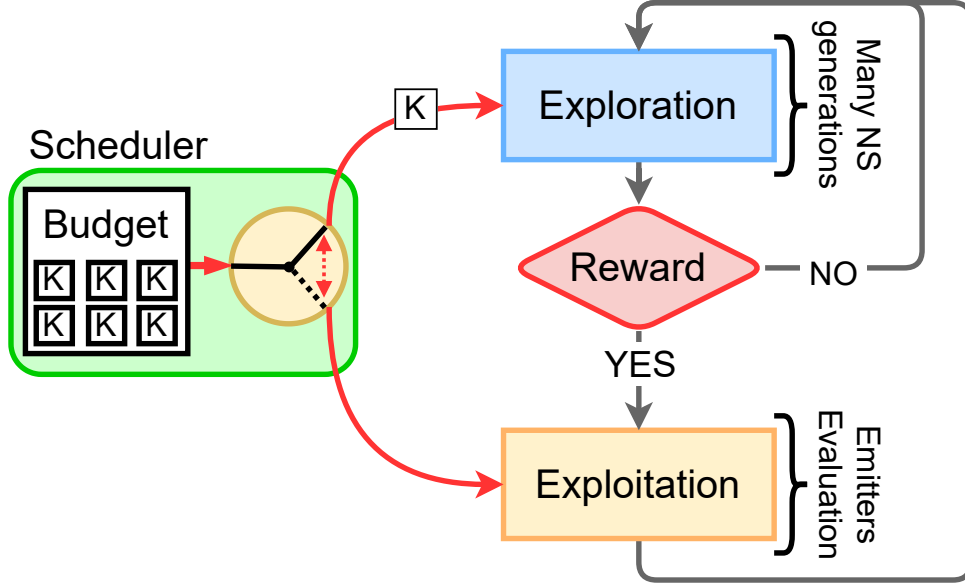
**SERENE** disentangles the exploration of the behavior space  $\mathcal{B}$  from the exploitation of the reward through a two-steps process. In the first phase, called *exploration phase*,  $\mathcal{B}$  is explored by performing **NS**. As per equation (2.5), the policies  $\theta_i$  found during exploration are assigned a behavior descriptor  $\phi(\theta_i)$ . A policy obtaining a reward means that its  $\phi(\theta_i)$  belongs to the subspace of rewarding behaviors  $\mathcal{B}_{\text{Rew}} \subseteq \mathcal{B}$ . It is in this subspace that the exploitation of the reward happens. This is done in the second phase, called *exploitation phase*, in which emitters are initialized using the rewarding policies found in  $\mathcal{B}_{\text{Rew}}$  during exploration. During the exploitation phase the most rewarding policies are stored to be returned as result of the algorithm. Moreover, the particularly novel policies found by the emitters are stored together with the policies found during the exploration process.

By launching emitters only in the neighborhoods of the reward areas, **SERENE** keeps the exploitation of the reward separated from the exploration of the search space. This results in taking the best of both worlds: the exploration power of **NS** and the focused exploitation of reward-based algorithms.

The exploitation and exploration phases are alternated repeatedly through a meta-scheduler. This scheduler divides a total evaluation budget  $Bud$  in smaller chunks of size  $K_{Bud}$  and assigns them to either one of the two phases. The whole process is illustrated in Figure 5.1 and described in Algorithm 4.

To keep track of policies generated during the different phases, **SERENE** uses the following buffers and containers:

- *novelty archive*  $\mathcal{A}_{\text{Nov}}$ : a repertoire of the novel policies found during the *exploration phase*, and returned as first output of **SERENE**;
- *reward archive*  $\mathcal{A}_{\text{Rew}}$ : a repertoire of rewarding policies found during the *exploitation phase*, returned as second output of **SERENE**;
- *candidates emitter buffer*  $\mathcal{Q}_{\text{Cand\_Em}}$ : a buffer containing the rewarding policies  $\phi(\theta_i) \in \mathcal{B}_{\text{Rew}}$  found during the *exploration phase* and used in the *exploitation phase* to initialize emitters;
- *emitter buffer*  $\mathcal{Q}_{\text{Em}}$ : a buffer containing all the initialized emitters to be evaluated during the *exploitation phase*;



**Figure 5.1:** *SERENE* consists of two exploration and exploitation processes, controlled by a scheduler. The exploration process searches for novel solutions through Novelty Search. The exploitation process uses emitters to optimize the rewards discovered during exploration. The scheduler alternates between the two processes by splitting the total evaluation budget into chunks of size  $K$  to assign to either of them.

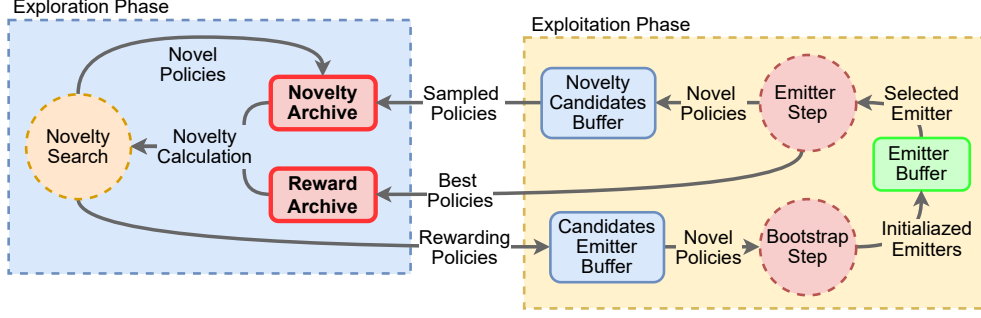
---

**Algorithm 4: SERENE**

---

**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  $N_Q$ ;  
**RESULT:** Novelty archive  $\mathcal{A}_{Nov}$ , rewarding archive  $\mathcal{A}_{Rew}$ ;  
 $\mathcal{A}_{Nov} = \emptyset$ ;  $\mathcal{A}_{Rew} = \emptyset$ ;  
 $\mathcal{Q}_{Em} = \emptyset$ ;  $\mathcal{Q}_{Cand\_Nov} = \emptyset$ ;  $\mathcal{Q}_{Cand\_Em} = \emptyset$ ;  
 $\Gamma_0 \leftarrow M$  policies from  $\Theta$ ;  
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  $b_i = \phi(\theta_i) \in \mathcal{B}$ ,  $\forall \theta_i \in \Gamma_0$ ;  
  **end**  
   $ExplorationPhase(K_{Bud}, m, \sigma, \mathcal{A}_{Nov}, \mathcal{Q}_{Cand\_Em}, \Gamma_g, N_Q)$ ;  
  **if not**  $\mathcal{Q}_{Cand\_Em} == \emptyset$  **or not**  $\mathcal{Q}_{Em} == \emptyset$  **then**  
     $ExploitationPhase(K_{Bud}, \mathcal{Q}_{Cand\_Em}, \lambda, m, \mathcal{Q}_{Em}, \mathcal{A}_{Nov}, \mathcal{A}_{Rew}, M_{\mathcal{E}})$ ;  
  **end**  
**end**

---



**Figure 5.2:** Overview of the sets used by *SERENE* to keep track of the explored areas and the initialized emitters. Highlighted in red are the two archives returned as final result of the algorithm execution.

- *novelty candidates buffer*  $\mathcal{Q}_{\text{Cand\_Nov}}$ : a buffer containing the most novel policies found by the emitter. Each emitter has its own instance of this buffer and the policies in it are sampled for addition to the novelty archive  $\mathcal{A}_{\text{Nov}}$  once the emitter is terminated.

A high-level overview of how these sets interact during the two phases is given in Figure 5.2, and a more detailed description is proposed in the two following subsections.

*SERENE* uses multiple populations during each process of the search. They are listed here for clarity:

- $\Gamma_g$ : population of size  $M$  at generation  $g$ . Used by *NS* during the exploration phase;
- $\Gamma_g^m$ : offspring population of size  $m \times M$  at generation  $g$ . Used by *NS* during the exploration phase, it is generated by spawning  $m$  agents from each policy  $\theta_i \in \Gamma_g$ ;
- $P_\gamma$ : emitter population of size  $M_\mathcal{E}$  at the emitter generation  $\gamma$ . Used by the emitter during the exploitation of the reward;
- $P_\gamma^m$ : emitter offspring population of size  $m \times M_\mathcal{E}$  at the emitter generation  $\gamma$ . Used by the emitter during the exploitation of the reward, it is generated by spawning  $m$  agents from each policy  $\tilde{\theta}_i \in P_\gamma$ .

### 5.3.1 Exploration phase

*SERENE* starts by generating an initial population  $\Gamma_0$  of size  $M$ . This is done by sampling the parameters of the population’s policies  $\theta_j$  from a normal distribution  $\mathcal{N}(0, I)$ . The population is used to explore the behavior space  $\mathcal{B}$  through *NS*. At each generation  $g$ , a mutation operator generates  $m$  new policies  $\theta_j^i$  (offspring) from each of the policies  $\theta_j \in \Gamma_g$ :

$$\forall j, i \in \{1, \dots, M\} \times \{1, \dots, m\}, \theta_j^i = \theta_j + \epsilon, \quad \text{with } \epsilon \sim \mathcal{N}(0, \sigma I). \quad (5.1)$$



The resulting offspring population  $\Gamma_g^m$ , of size  $m \times M$ , is then evaluated to obtain the behavior descriptors  $\phi(\theta_j^i) = b_j^i \in \mathcal{B}$ , used to calculate the novelty of  $\Gamma_g$  and  $\Gamma_g^m$  according to Eq. (5.2):

$$\eta(\theta_i) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(b_i, b_j) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(\phi(\theta_i), \phi(\theta_j)), \quad (5.2)$$

The novelty is then used to generate the next generation population  $\Gamma_{g+1}$  by taking the most novel policies from the current population and the offsprings. At the same time,  $N_Q$  policies among the offsprings are uniformly sampled to be added to the *novelty archive*  $\mathcal{A}_{\text{Nov}}$ . Finally, all the policies for which  $r_i > 0$  are stored in the *candidates emitters buffer*  $\mathcal{Q}_{\text{Cand\_Em}}$ . The process just described is detailed in Algorithm 5.

The exploration phase is executed for the  $K_{\text{Bud}}$  evaluation steps in the given budget chunk, where each evaluation step corresponds to one policy evaluation. Once the chunk is depleted, the scheduler assigns the next chunk to the *exploitation phase* only if  $\mathcal{Q}_{\text{Cand\_Em}} \neq \emptyset$ . On the contrary, if the buffer is empty, i.e. no new reward has been found during exploration, another *exploration phase* is performed. This means that in the worst case scenario where no reward can be discovered, i.e.  $\mathcal{B}_{\text{Rew}} = \emptyset$ , SERENE performs exactly like NS.

---

**Algorithm 5:** SERENE Exploration Phase

---

**INPUT:** budget chunk  $K_{\text{Bud}}$ , 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  $N_Q$ ;

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

- Generate offspring  $\Gamma_g^m$  from population  $\Gamma_g$ ;
- Evaluate  $\theta_i$ ,  $\forall \theta_i \in \Gamma_g^m$ ;
- Calculate  $b_i = \phi(\theta_i) \in \mathcal{B}$ ,  $\forall \theta_i \in \Gamma_g^m$ ;
- Calculate  $\eta(\theta_i) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(b_i, b_j)$ ,  $\forall \theta_i \in \Gamma_g^m \cup \Gamma_g$ ;
- $\mathcal{A}_{\text{Nov}} \leftarrow N_Q$  samples from  $\Gamma_g^m$ ;
- if**  $\phi(\theta_i) \in \mathcal{B}_{\text{Rew}}$  **then**
  - |  $\mathcal{Q}_{\text{Cand\_Em}} \leftarrow \theta_i$
- end**
- Generate  $\Gamma_{g+1}$  from most novel  $\theta_i \in \Gamma_g^m \cup \Gamma_g$ ;

**end**

---

### 5.3.2 Exploitation phase

The *exploitation phase* consists of two sub-steps: the *bootstrapping step*, in which the policies in the candidates emitter buffer  $\mathcal{Q}_{\text{Cand\_Em}}$  are used to initialize and bootstrap emitters, and the *emitter step*, in which the initialized emitters are evaluated.

## Bootstrap step

During this step, emitters are initialized from the rewarding policies  $\theta_i$  in the candidates emitter buffer  $\mathcal{Q}_{\text{Cand\_Em}}$ , and their potential for reward improvement evaluated. This insures that only emitters capable of improving the rewards are considered for full evaluation, reducing wasted evaluation budget. The policies used to initialize the emitters are selected according to their novelty with respect to the reward archive  $\mathcal{A}_{\text{Rew}}$ . This enables SERENE to focus on less explored areas of the rewarding behavior space  $\mathcal{B}_{\text{Rew}}$ . The whole bootstrapping phase lasts  $K_{\text{Bud}}/3$  evaluations.

As discussed in Section 5.2, an emitter is an instance of a *reward-based algorithm*. Contrary to the previous work of Fontaine et al. [55] and Cully [56], this work does not use estimation-of-distribution algorithms like CMA-ES [162] but rather an elitist reward-based EA. The reason behind this choice is that, as the name suggests, estimation-of-distribution algorithms work by estimating a probability distribution, usually a gaussian, from where the policies for the next generation are selected. Having to estimate a distribution requires the estimation of a covariance matrix  $\Sigma$  whose reliability strongly depends on the ratio between the dimension of the parameter space  $\Theta$  and the size of the population. If the size of population used to estimate the matrix is smaller than the dimension of the parameter space, the estimation of  $\Sigma$  is not reliable. CMA-ES circumvents the issue by using information from previous generations to calculate  $\Sigma$  through the evolution path. While stabilizing  $\Sigma$ , having to wait for multiple generations leads to a less efficient use of the evaluation budget. Hence, the emitters used in this work are based on an *elitist evolutionary algorithm* not requiring the estimation of any distribution. Conversely, the population is composed with the most rewarding policies from the previous generation’s population and offspring, while the offspring are generated by mutating the parents according to equation 5.1.

An emitter  $\mathcal{E}_i$  based on this algorithm consists of:

- a population  $P$  containing  $M_{\mathcal{E}}$  policies  $\tilde{\theta} \in \Theta$ ;
- a population of offspring  $P^m$  of size  $m \times M_{\mathcal{E}}$ ;
- a generation counter  $\gamma$ ;
- a tracker for the maximum reward found so far  $R_{\gamma}$ ;
- an improvement measure  $I(\cdot)$ ;
- a novelty measure  $\eta_i$  equal to the novelty of the policy used to initialize the emitter;
- a *novelty candidate buffer*  $\mathcal{Q}_{\text{Cand\_Nov}}$ .

The emitter  $\mathcal{E}_i$  is initialized from a policy  $\theta_i$  in the candidates emitter buffer by sampling its initial population  $P_0$  from the distribution  $\mathcal{N}(\theta_i, \sigma_i I)$ . To keep

the emitter’s exploration local and prevent overlapping with the search space of possible nearby emitters,  $\sigma_i$  is initialized as:

$$\sigma_i = \frac{\min_j (\text{dist}(\theta_i, \theta_j))}{3}, \quad \forall \theta_j \in \Gamma_g^m \cup \Gamma_g. \quad (5.3)$$

This shapes  $\mathcal{N}(\theta_i, \sigma_i I)$  such that all other  $\theta_j$  are at least 3 standard deviation away from its center. Once  $\mathcal{E}_i$  has been initialized, its potential is evaluated by running it for  $\lambda$  generations and calculating its *emitter improvement*  $I(\mathcal{E}_i)$ . This improvement is defined as the difference between the average rewards obtained during the most recent and the initial generations of the emitter:

$$I(\mathcal{E}_i) = \frac{1}{\lambda M_{\mathcal{E}}} \left( \sum_{\gamma=T-\lambda/2}^T \sum_{j=0}^{M_{\mathcal{E}}} r_{(\gamma,j)} - \sum_{\gamma=\gamma_0}^{\lambda/2} \sum_{j=0}^{M_{\mathcal{E}}} r_{(\gamma,j)} \right). \quad (5.4)$$

Here  $T$  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 the exploitation phase; it is always  $\gamma_0 = 0$  for an emitter in the bootstrap step. If  $I(\mathcal{E}_i) \leq 0$ , the chances for the emitter to find better solutions than the initial ones are low, so it is not worth allotting more budget to its evaluation. On the contrary,  $I(\mathcal{E}_i) > 0$  means that the emitter has high potential for improvement. Thus all the initialized emitters for which  $I(\mathcal{E}_i) > 0$  are added to the *emitter buffer*  $\mathcal{Q}_{\text{Em}}$  for further evaluation.

### Emitter step

The initialized emitters in the *emitter buffer*  $\mathcal{Q}_{\text{Em}}$  are run during this step. It starts by calculating the pareto front between the improvement  $I(\mathcal{E}_i)$  and the novelty  $\eta(\mathcal{E}_i)$  of each of the emitters  $\mathcal{E}_i$  in the emitter buffer. The emitter to run is then randomly sampled from the front of the *non-dominated* emitters. Using both the novelty and the fitness to select which emitter to run allows [SERENE](#) to focus both on the less explored and most promising areas of  $\mathcal{B}_{\text{Rew}}$ .

The policies  $\tilde{\theta}_j$  generated by an emitter can be stored either for the reward they achieve or for their novelty. At every generation  $\gamma$  all the policies  $\tilde{\theta}_j$  in the current population with a reward  $r(\tilde{\theta}_j) > R_{\gamma-1}$  are added to the reward archive  $\mathcal{A}_{\text{Rew}}$ . Additionally, the policies  $\tilde{\theta}_j$  with a novelty higher than the emitter novelty  $\eta_i$  are stored into the emitter’s *novelty candidates buffer*  $\mathcal{Q}_{\text{Cand\_Nov}}$ .

The emitter  $\mathcal{E}_i$  is run until either the given budget chunk is depleted or a termination condition is met. In the first case, [SERENE](#) recalculates the improvement  $I(\mathcal{E}_i)$  from the beginning of the *emitter phase* and assigns the next budget chunk to the *exploration phase*. On the contrary, if a termination condition is met,  $\mathcal{E}_i$  is discarded and another emitter to evaluate is sampled from the Pareto front. There can be multiple termination conditions depending on the algorithm used as emitter. A CMA-ES based emitter can use all the termination criteria listed by Hansen in Appendix B.3 in its tutorial on CMA-ES [162]. The one used in this work is also inspired from Hansen’s work [162], namely the *stagnation criterion*, which stops the emitter when there is

---

**Algorithm 6: SERENE** 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  $\sigma_i$ ;
    Initialize:  $\mathcal{E}_i$ ,  $\mathcal{Q}_{Cand\_Nov}^i = \emptyset$ , and  $P_0$ ;
    for  $\gamma \in \{0, \dots, \lambda\}$  do
        if  $P_0$  then
            Evaluate  $\tilde{\theta}_j, \forall \tilde{\theta}_j \in P_0$ ;
        end
        Generate offspring population  $P_\gamma^m$  from  $P_\gamma$ ;
        Evaluate  $\tilde{\theta}_j, \forall \tilde{\theta}_j \in P_\gamma^m$ ;
        Generate  $P_{\gamma+1}$  from best  $\tilde{\theta}_j \in P_\gamma^m \cup P_\gamma$ ;
    end
    Calculate  $I(\mathcal{E}_i)$ ;
    if  $I(\mathcal{E}_i) > 0$  then
         $\mathcal{Q}_{Em} \leftarrow \mathcal{E}_i$ ;
    end
end
/* Emitters step */
Calculate pareto fronts in  $\mathcal{Q}_{Em}$ ;
while  $2/3K_{Bud}$  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  $P_\gamma$ ;
        Evaluate  $\tilde{\theta}_j, \forall \tilde{\theta}_j \in P_\gamma^m$ ;
         $\mathcal{A}_{Rew} \leftarrow \tilde{\theta}_j, \forall \tilde{\theta}_j \in P_\gamma^m \mid r(\tilde{\theta}_j) > R_\gamma$ ;
         $\mathcal{Q}_{Cand\_Nov}^i \leftarrow \tilde{\theta}_j, \forall \tilde{\theta}_j \in P_\gamma^m \mid \eta(\tilde{\theta}_j) > \eta_i$ ;
        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
             $\mathcal{A}_{Nov} \leftarrow N_Q$  samples from  $\mathcal{Q}_{Cand\_Nov}^i$ ;
            Discard emitter  $\mathcal{E}_i$ ;
        end
    end
end
end
```

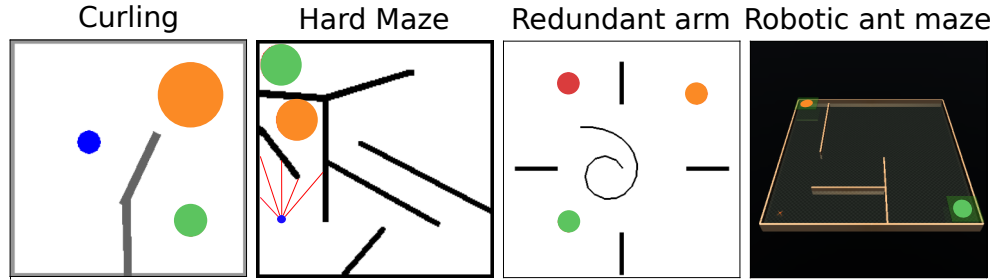
---

no more improvement on the reward. To do so the history of the rewards is tracked over the last  $120 + 20 * n / M_{\mathcal{E}}$  emitter’s generations, where  $n$  is the size of the parameter space  $\Theta$  and  $M_{\mathcal{E}}$  is the emitter’s population size. The emitter is terminated if either the maximum or the median of the last 20 rewards is not better than the maximum or the median of the first 20 rewards. Before starting the new emitter evaluation,  $N_Q$  policies from the terminated emitter’s *novelty candidates buffer*  $\mathcal{Q}_{\text{Cand\_Nov}}$  are uniformly sampled to be added to the novelty archive  $\mathcal{A}_{\text{Nov}}$ . In addition to saving particularly novel solutions as part of the final result, this prevents the exploration phase from re-exploring areas covered by emitters during the exploitation phase.

The whole *exploitation phase* is detailed in Algorithm 6.

## 5.4 Experiments

This section tests if SERENE can efficiently deal with sparse reward settings, finding all disjoint reward areas, and optimizing the reward in each of them. For the evaluation, the following four sparse rewards environments - illustrated in Fig. 5.3 - are considered:



**Figure 5.3:** Testing environments: *Curling*, *HardMaze*, *Redundant arm*, *Robotic ant maze*.

### Curling

A two **degrees of freedom (DoF)** robotic arm [163] controlled by a 3 layers NN with each layer of size 5. The arm has to push the blue ball into one of the two goal areas shown in orange and green. A reward is provided only if the ball stops in one of the two areas. Moreover, the closer the ball is to the center of the reward area, the higher the reward is. The controller takes as input a 6-dimensional vector containing the ball pose  $(x, y)$ , and the two joints angles and velocities. The output of the controller is the speed of each joint at the next timestep. The size of the parameter space  $\Theta$  is 94, and each policy is run in the environment for 500 timesteps.

### Hardmaze

Inspired by the Hardmaze environment introduced by Lehman and Stanley [34], it consists of a two-wheeled robot, in blue, whose task is to navigate

the maze and reach either one of the green and orange areas. The reward is only given if the robot stops in one of the two areas, and is higher the closer the robot is to its center. The robot is controlled by a 2-layers **NN** with each layer of size 5. The controller takes as input the reading of the 5 distance sensors mounted on the robot; shown in red in Figure 5.3. Its output is the 2-dimensional vector containing the speed of the 2 wheels at the next timestep. The size of the parameter space  $\Theta$  is 63, and each policy is run in the environment for 2000 timesteps.

### Redundant arm

A 20-DoF robotic arm in which the arm’s end-effector has to reach one of the 3 colored goal areas. The reward is maximal in the center of the areas, and the arm is controlled by a **NN** with 2 layers of size 5. This environment is based on the one introduced by Loviken and Hemion [164]. The controller takes as input the 20-dimensional vector of each joint’s position, and outputs the 20-dimensional joint’s torque vector. The size of the parameter space  $\Theta$  is 228, and each policy is run in the environment for 100 timesteps.

### Robotic ant maze

Based on the setup introduced by Cideron et al. [135], it consists of a 4-legged robotic ant in a maze. There are two goal areas and the task is for the ant to navigate the maze and reach the center of one of them. The robot is controlled by a 3-layers **NN**, with each layer of size 10. The input of the controller is the 29-dimensional observation returned by the environment at each step, while its output is the 8-dimensional joint’s torque control. The size of the parameter space  $\Theta$  is 574, and each policy is run in the environment for 3000 timesteps.

For all environments, the reward is given only if inside the reward area, and as a continual value in the  $[0, 1]$  range. The reward varies with the distance to the center of the area and is highest directly at the center. It can be expressed as:

$$r(\theta) = \begin{cases} 0, & \text{if } d_r > radius \\ \frac{radius-d_r}{radius}, & \text{if } d_r \leq radius \end{cases}$$

where  $d_r$  is the distance from the center and *radius* is the radius of the reward area.

Structuring the reward in this way allows to have a reward gradient once inside the reward area, providing the possibility to improve on it and to highlight the advantages provided by the emitters. Were the reward to be binary, there would be no need to use emitters because there is no improvement to be done on the reward once discovered, thus vanilla **NS** would be enough.

## Baselines

**SERENE** is compared against 5 different baselines:

- **NS**[34]: vanilla **NS**, that performs pure exploration and does not attempt to improve on the reward;
- **MOO-NR**[115]: a multi-objective evolutionary algorithm optimizing both the novelty and the reward;
- **CMA-ME**[55]: the original algorithm introducing emitters that combines **ME** with emitters over a  $50 \times 50$  grid covering the behavior space of all environments. Among the various emitters proposed with **CMA-ME** [55], the “optimizing” emitter was selected;
- **ME**[43]: vanilla MAP-Elites that uses a  $50 \times 50$  grid to cover the behavior space of every environment;
- **RND**: pure random search in which no selection happens, and every policy is sampled from a normal distribution  $\mathcal{N}(0, I)$ .

For each experiment the total evaluation budget is  $Bud = 500000$ , with the chunk size set to  $K_{Bud} = 1000$ . The population size is  $M = 100$ , and each policy generates  $m = 2$  offspring. The mutation parameter is set to  $\sigma = 0.5$ , while the number of policies uniformly sampled to be added to the novelty archive is  $N_Q = 5$ . **SERENE** uses an emitter population size of  $M_{\mathcal{E}} = 6$ , with a bootstrap phase for each emitter of  $\lambda = 6$  generations. The implementation of **CMA-ME** uses the same parameters used in Fontaine et al. [55]: 15 emitters, each one with a population size of 37. In every experiment, the policies parameters are clipped in the  $[-5, 5]$  range. Finally, the statistical results are computed over 15 runs for each experiment.

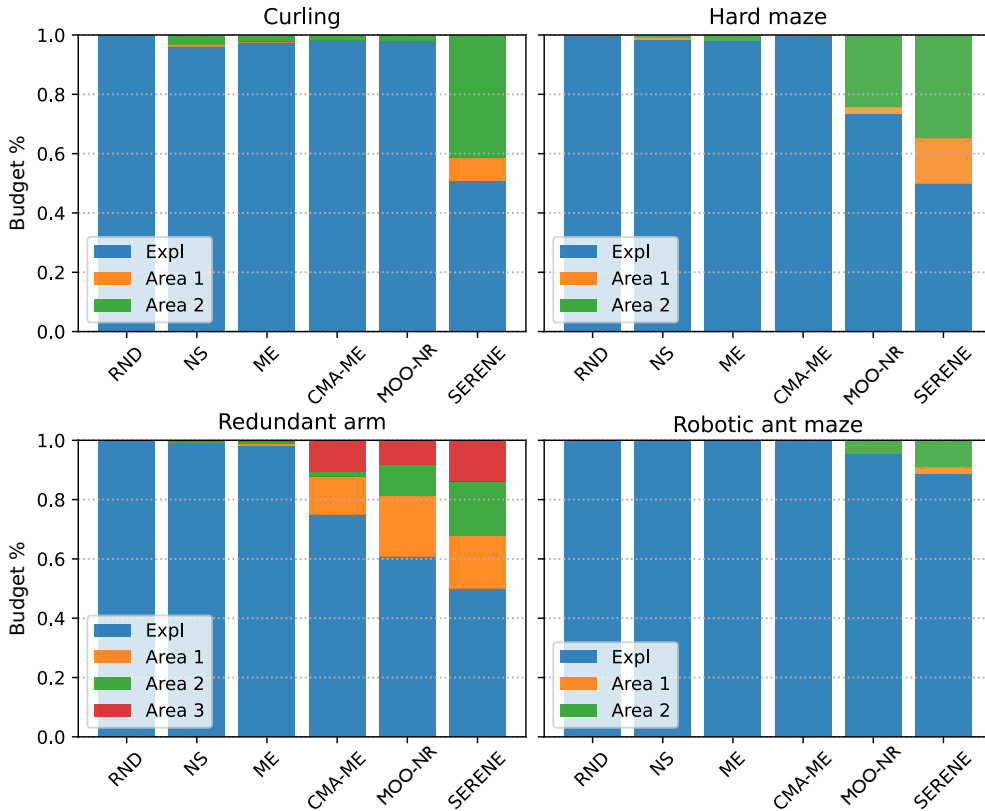
## 5.5 Results

This section discusses the results obtained during the experiments.

### 5.5.1 Budgeting

Balancing the exploration of the search space and the exploitation of the reward is an aspect of paramount importance for reward-based algorithms. Even more so in sparse reward environments. This balance can be studied by analyzing the amount of evaluation budget dedicated to either one of the two aspects. The exploration budget consists of all the evaluated policies that did not get any reward. On the contrary, the exploitation budget is obtained by counting all the evaluated policies that collected some reward from one of the reward areas.

As Figure 5.4 shows, **SERENE** has a more balanced budget split between exploration (in blue) and exploitation (other colors) compared to the other baselines. In situations in which exploration is harder, a bigger part of the

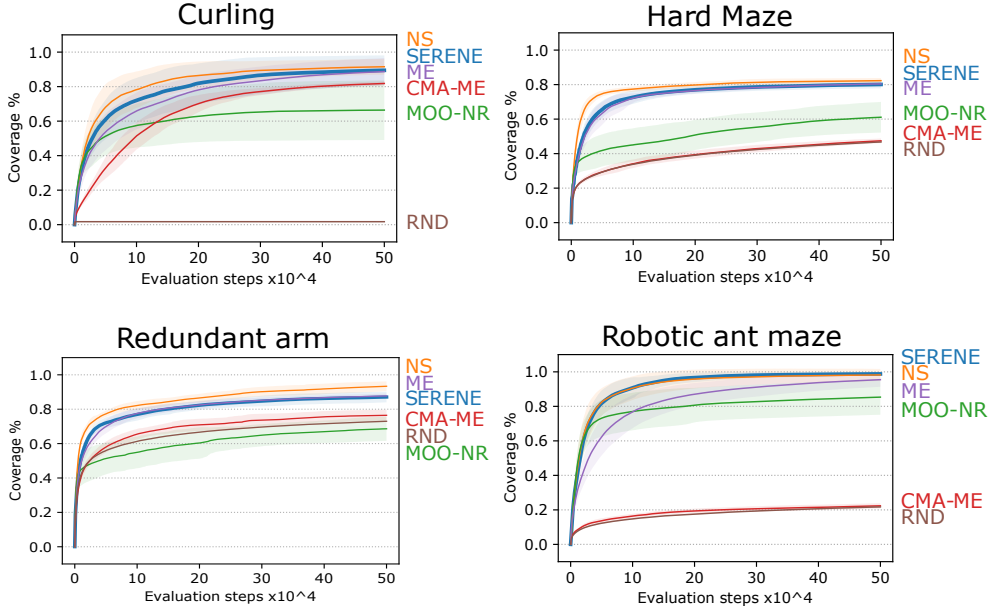


**Figure 5.4:** Average budget percentage between the exploration of the search space (in blue) and the exploitation of each reward areas (other colors).

budget is assigned to exploration rather than exploitation of the reward. This is the case for the robotic ant maze environment. Additionally, due to the way emitters are selected, the algorithm can shift its exploitation focus among the different reward areas. Figure 5.4 shows that most of **SERENE**'s exploitation budget is assigned to the green reward area in the Curling, Hard maze and Robotic ant maze environments. As it can be seen in Figure 5.3, this area is more difficult to discover and to reach with respect to the orange area. This makes the exploitation of the orange reward area faster, having both the novelty and the improvement go to zero rapidly. On the contrary, the novelty of the harder to reach green area remains higher for longer, making **SERENE** more likely to select emitters focused on it. The effect can also be seen in Figure 5.6, where the reward for area 1 quickly reaches higher values compared to the one of reward area 2. At the same time, in the Redundant arm environment where the 3 reward areas are equally easy to discover and to reach, this effect is less present and the exploitation budget is more evenly split between them. The ability to switch its focus is similar to intrinsic motivation based methods [165, 166] and allows **SERENE** to reach high rewards in all reward areas. Other baselines exhibit a less balanced distribution of the evaluation budget, as they do not explicitly separate exploration from exploitation.



## 5.5.2 Exploration



**Figure 5.5:** Average coverage with respect to the given evaluation budget. The shaded areas represent one standard deviation.

Performing good exploration in situations of sparse rewards is fundamental in order to discover all the possible rewarding areas of the search space. In the experiments, the exploration capacity of each of the tested algorithms was measured through the *coverage metric* [43, 48]. It is evaluated by discretizing the search space in a  $50 \times 50$  grid and calculating the percentage of cells occupied by the policies found during the search. This metric does not include any measure of the performance of the solutions in the cells.

The plots in Figure 5.5 show that **SERENE** can perform exploration with an efficiency comparable to **NS**, notwithstanding the lower budget assigned to exploring the search space. At the same time, Figure 5.5 shows that the final coverage obtained by **ME** is similar to the one of **NS** and **SERENE**.

On the contrary, although based on **ME**, CMA-ME results are more variable across all environments, and exhibit lower exploration compared to **ME**. This effect is likely due to the reliance on emitters for exploration, leading to more local exploration in the parameter space  $\Theta$ . It can prove useful in environments like Curling or Redundant arm, where a small change in parameters leads to big behavioral changes, increasing the probability of finding a reward. On the contrary, environments like Hard Maze or Robotic ant maze in which this does not happen can prove more challenging to explore.

At the same time, the exploration performance of **MOO-NR** is poor. In the Redundant arm environment, exploration is even lower than the random search baseline. This result is likely due to the multi-objective approach of optimizing both novelty and reward through Pareto fronts. With time, finding more novel parts of the environment becomes increasingly more difficult. This

is due to the rewarding solutions dominating all policies outside of the reward areas that have no reward. The non-rewarding solutions - that can foster exploration towards unexplored areas of the search space - will be then less likely to be selected. This biases the algorithm towards the exploitation of the already discovered rewards rather than the exploration of the search space.

### 5.5.3 Exploitation

Fig. 5.6 shows the average maximum reward achieved by the algorithms in the reward areas of all environments. Emitters solely focusing on exploiting the reward allow **SERENE** to reach almost the maximum reward on the easiest to reach reward areas in less than  $10^5$  evaluations. High rewards are also achieved on the harder to reach areas, even if the required time is higher. On the contrary, **ME** improves on the reward at a much slower pace. This is likely due to the random selection of policies from the archive to generate new policies. In a sparse reward environment in fact, the probability of selecting a rewarding policy is proportional to the ratio between the rewarding and non-rewarding areas. The sparser the reward is, i.e. the smaller the reward area is, the lower the probability of selecting a rewarding policy from the archive is, and the slower the exploitation gets. A similar trend is exhibited by CMA-ME: even if able to reach high rewards on the discovered reward areas, it is slow in its optimization. At the same time, even **NS** reached high rewards on almost all environments, but without any explicit reward optimization it did not exploit the reward areas to the maximum. The multi-objective approach **MOO-NR** 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.

### 5.5.4 Final archive distribution

Fig. 5.7 shows the distribution of the behaviors of the policies in the final archive. Each point represents a different policy. In blue are the policies that do not get any reward, thus considered *exploratory*, while in orange are rewarding policies, considered *exploitative*. For **SERENE** the *exploratory policies* are the ones in the *novelty archive*  $\mathcal{A}_N$ , while the *exploitative policies* are the ones in the *rewarding archive*  $\mathcal{A}_R$ .

From the figure, it is possible to see that **SERENE** covers the search space well, in a fashion similar to **NS**, with the exception of the reward areas, where **SERENE**'s emitters allow to have a denser exploration (and thus exploitation of the reward). A similar dense exploration of the reward areas is performed by **MOO-NR** but at the cost of exploration. This is likely due to the fact that once a reward is discovered, the reward scales more than the novelty, making the algorithm focus more on the reward improvement than the exploration. In the Curling and Robotic ant maze environments this effect is so strong that it prevents the discovery of all the reward areas. At the same time, and contrary to **NS** based methods, both **ME** and CMA-ME can explore the search space in

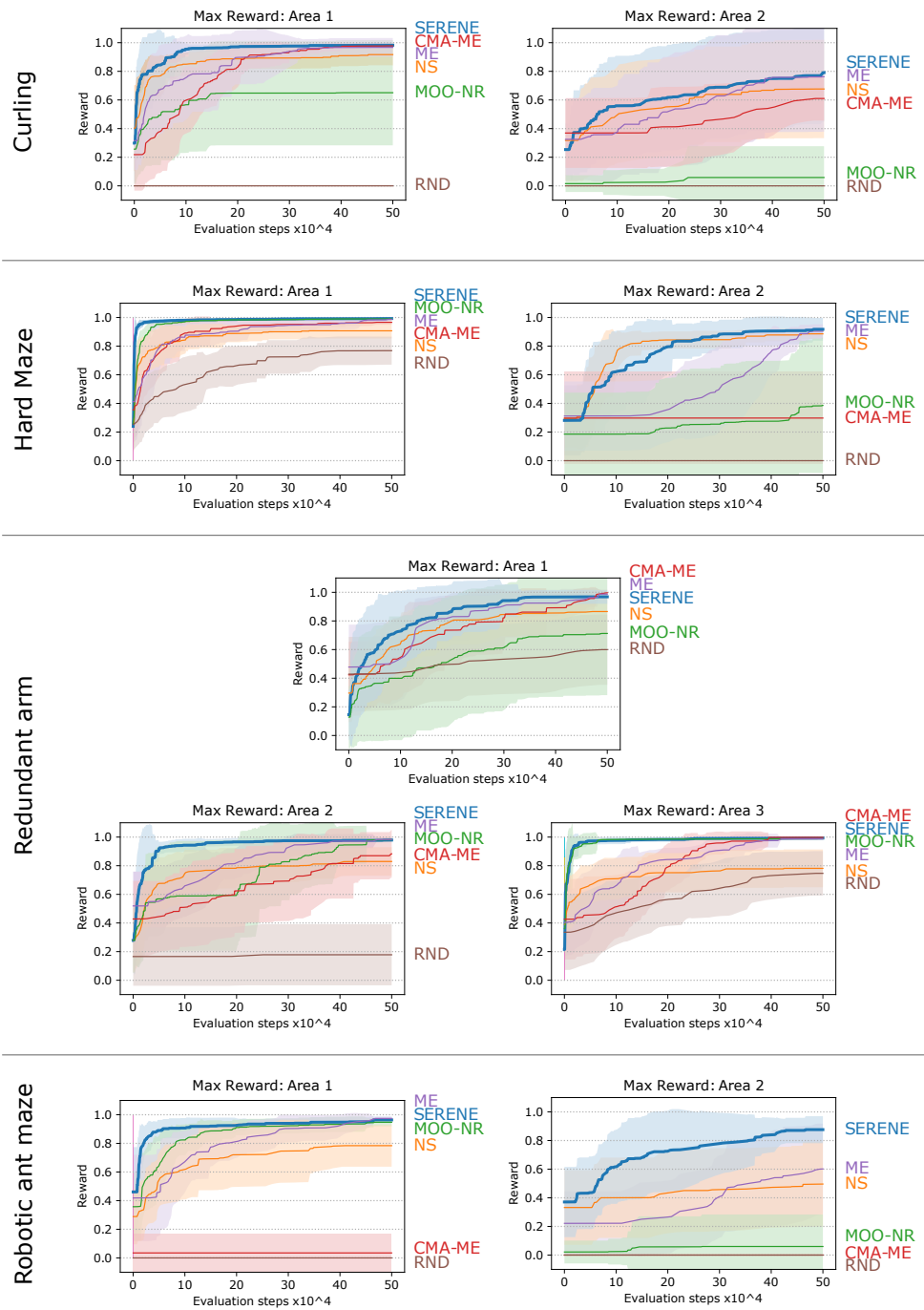
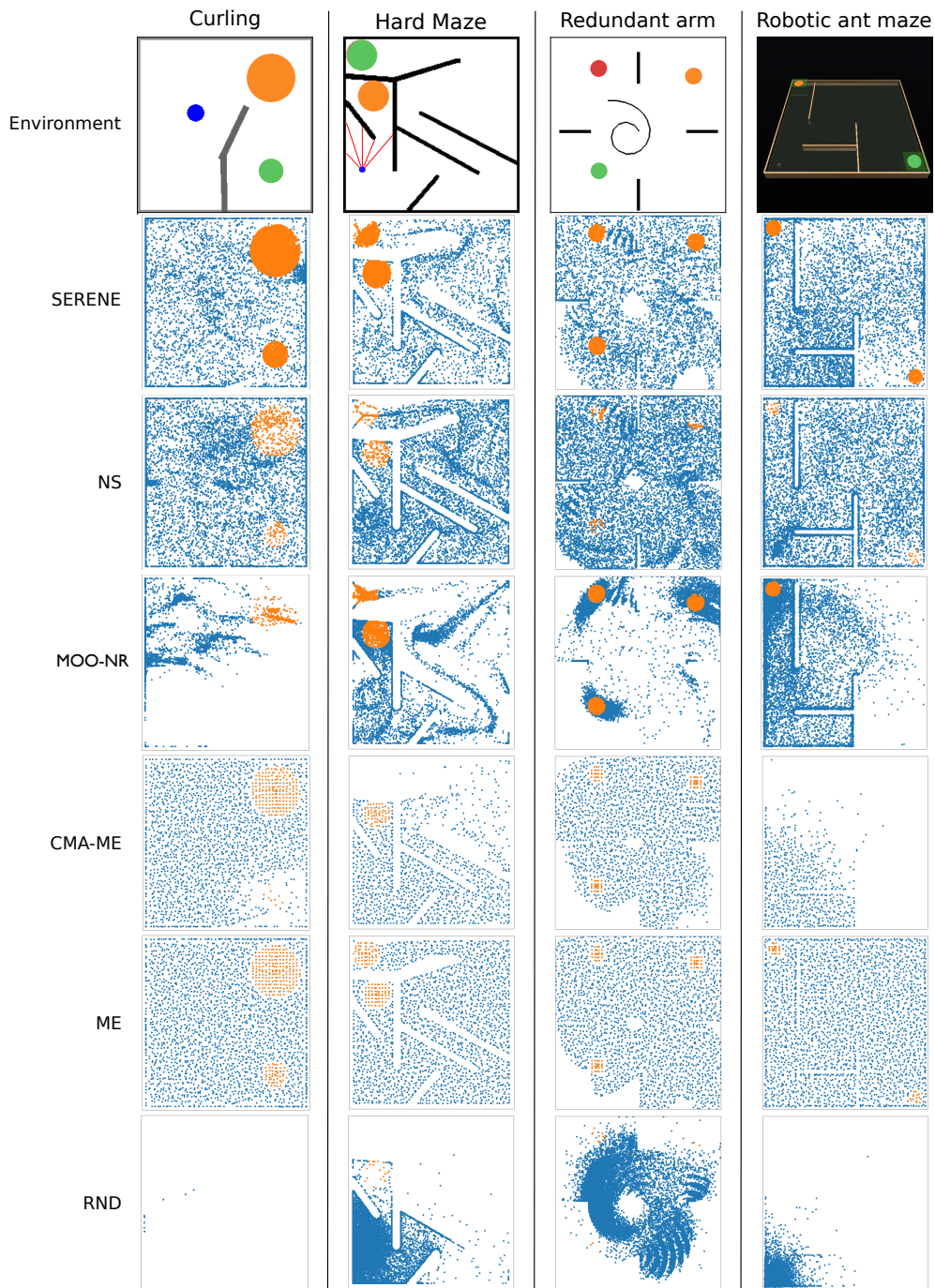


Figure 5.6: Average maximum reward reached in all the reward areas. The shaded areas represent one standard deviation.



**Figure 5.7:** *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 **SERENE** in blue are the policies in the novelty archive and in orange the policies in the reward archive.*

a much more uniform fashion, thanks to the discretization of the search space performed by the algorithm. While this discretization is enough to push the exploration to cover the whole space for [ME](#), this is not the case for CMA-ME, as it can be seen especially in the hardest to explore environment: the Robotic ant maze. This is the result of relying on emitters for both reward exploitation and exploration. This strategy forces emitters to both diversify the behavior of the discovered policies, trying to cover as much search space as possible, and optimize the policies performances, focusing on a very narrow area of the whole search space.

## 5.6 Conclusion

In this chapter [SERENE](#) was introduced, a method that efficiently deals with sparse reward environments by augmenting [NS](#) with emitters. Contrary to similar methods using emitters, [SERENE](#) keeps exploration and exploitation of the reward as two distinct processes. Exploration is carried out by taking advantage of [NS](#) to discover all the reachable reward areas. These areas are then exploited by using local instances of population-based optimization algorithms called emitters. By using a meta-scheduler, [SERENE](#) can automatically assign the evaluation budget to either exploration or exploitation. This is advantageous also in situations in which no reward is present: in the absence of reward to exploit, [SERENE](#) performs exactly like [NS](#).

[SERENE](#) has been tested on four different sparse reward environments, reaching high performances on all of them. At the same time, many kind of emitters can be used to address different kind of problems [55, 56]. The implementation of various types of reward-based algorithms as emitters and their combination can prove an exciting line of work to extend the current method to new domains.

Notwithstanding these encouraging results, the method still suffers from the same limitations as other [QD](#) methods, and first and foremost from the prior hand-design of the behavior space  $\mathcal{B}$ . The next chapter will present a method that can learn a behavior descriptor useful for exploration while exploiting any discovered reward through emitters. This is done by building on the ideas introduced through [TAXONS](#) and [SERENE](#).



---

**Chapter content**

<b>6.1</b>	<b>Introduction</b>	<b>82</b>
<b>6.2</b>	<b>Method</b>	<b>83</b>
6.2.1	Policy Selection	84
6.2.2	Training of the autoencoder	84
6.2.3	Reward exploitation in a learned space	86
<b>6.3</b>	<b>Experiments</b>	<b>88</b>
<b>6.4</b>	<b>Results</b>	<b>92</b>
6.4.1	Exploration	92
6.4.2	Exploitation	94
6.4.3	Final archives distribution	94
6.4.4	Exploration ablation studies	96
6.4.5	Autoencoder training regime	100
6.4.6	Learned behavior space	103
<b>6.5</b>	<b>Conclusion</b>	<b>106</b>

---

## 6.1 Introduction

Previous chapters addressed two of the major shortcomings of **NS** when dealing with sparse reward systems. Chapter 3 introduced **TAXONS** to reduce the amount of prior information needed to be specified at design time. This is done by learning the behavior descriptor during the search in an autonomous way through an **AE**. Then, in chapter 5, the other limitation of **NS**, even more relevant in sparse rewards settings, was addressed: the inability to optimize any reward discovered during the search. When discovering a reward, in fact, the algorithm should be able to shift its focus on it in order to find the best possible solutions. **SERENE** does that through the use of emitters, by performing local search around any possible reward discovered. The two methods address separate but complementary limitations of **NS**. In this chapter, the two algorithms are brought together by introducing **STAX**. This algorithm uses **TAXONS** to perform exploration while learning a low-dimensional representation of the search space; the discovery of a reward triggers the instantiation of an emitter around the rewarding policy in order to locally explore



and exploit the reward. **STAX** performs the exploitation of the reward in the same way **SERENE** does: through emitters and through the alternating of the exploration and exploitation steps thanks to a meta-scheduler.

The advantages of this method are twofold: it removes the need of hand-designing a **BS** by directly learning a low-representation of the search space from high-dimensional observations of the policies behaviors. At the same time, it also removes the reward-related limitation of **NS**-based approaches by swiftly exploiting any reward discovered through emitters. All of this allows **STAX** to efficiently deal with sparse reward environments with minimum prior information required about the task at design time.

## 6.2 Method

As stated in the previous section, **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. This is done through a *meta-scheduler* whose task is to split the total evaluation budget  $Bud$  in small chunks of size  $K_{Bud}$  and assigning them to either one of the two sub-processes, in the same way the **SERENE** algorithm discussed in Chapter 5 does.

While **SERENE** and **STAX** perform *exploitation* in the same way, that is through emitters spawned around rewarding solutions, as shown in Alg. 6, the main difference between the two algorithms lies in the way *exploration* is performed. Rather than relying on **NS** to explore an hand-designed **BS**, as **SERENE** does, **STAX** takes advantage of **TAXONS** to learn a **BS** representation while performing the search in it. This reduces the amount of prior information needed to solve the task by removing the requirement of hand-designing the **BS**.

As explained in Chapter 3, **TAXONS** learns a **BS** through the use of an **AE** that 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  $\mathcal{B}$ , also called *outcome space*. The way the different spaces are connected through the **AE** can be defined as in Eq. (3.1), reported here in Eq. (6.1):

$$\begin{aligned} E : \mathcal{O} &\rightarrow \mathcal{F} \equiv \mathcal{B} \\ D : \mathcal{F} &\rightarrow \mathcal{O} \end{aligned} \tag{6.1}$$

where  $\mathcal{O}$  is the observation space,  $E$  is the encoder of the **AE** and the  $D$  is the decoder. There are two main differences between the exploration performed by **TAXONS** and **STAX**: the policy selection and the **AE**'s training regime.



### 6.2.1 Policy Selection

As discussed in Chapter 3, **TAXONS** drives the search for novel policies through two different metrics: *novelty* and *surprise*. The first one is calculated as the average distance between the policy  $\theta_i$  and the  $|J|$  closest other policies in the learned outcome space, as defined in Eq. (3.5). The *surprise* is calculated as the **AE**'s *reconstruction error* over the observations generated by the policy  $\theta_i$ , as defined in Eq. (3.6). A higher surprise on an outcome  $o_T$  implies that the **AE** has not seen that outcome very often, thus the area of the observation space  $\mathcal{O}$  close to  $o_T$  has not yet been properly explored. This means that by selecting policies whose outcome has higher surprise the algorithm can push towards more exploration.

**STAX** also uses these two metrics, but instead of using only the last observation  $o_T$  to calculate them, multiple observations sampled along the trajectory are used. As discussed in Chapter 4, this allows to remove the assumption of the last observation encoding enough information to describe the whole behavior of a policy  $\theta_i$ . This requires some modifications to the way the behavior descriptor is calculated with respect to **TAXONS**. **STAX** builds this descriptor by concatenating the **AE**'s low-dimensional representations of the sampled observations. Eq. (3.4) can then be rewritten as:

$$f(\theta_i) = [\dots, E(o_{t_k}), \dots, E(o_{t_K})], \quad (6.2)$$

where  $o_{t_k}$  is the observation generated by the policy  $\theta_i$  at time-step  $t_k$ .

At the same time, the surprise is calculated as the *sum* of the reconstruction errors over each observation. Meaning that Eq. (3.6) can be rewritten as:

$$s(\theta_i) = \sum_{k \in K} \|o_{t_k}^{(\theta_i)} - D(E(o_{t_k}^{(\theta_i)}))\|^2, \quad (6.3)$$

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

Other than the way novelty and surprise are calculated, **STAX** differs from **TAXONS** in the way these metrics are exploited. At each generation  $g$ , **TAXONS** randomly chooses only one between novelty and surprise to select the best policies. On the contrary, during **STAX**'s exploration step, the two metrics are combined through the NSGA-II multi-objective approach [115] detailed in Sec. 2.2.1. This means that the algorithm can optimize at the same time both the novelty and the surprise. The whole exploration process is shown in Alg. 7.

### 6.2.2 Training of the autoencoder

In **STAX** the exploration is driven thanks to the **AE**. This means that the way the **AE** is trained is fundamental. In order to meaningfully look for diversity in the learned feature space  $\mathcal{F}$  used as **BS**, the **AE** has to be trained on the data collected during the search for policies itself. This is done in a fashion similar to what described in Chapter 3 for the **TAXONS** algorithm, with a few differences. **TAXONS** built the dataset  $DS$  used to train the **AE** with

---

**Algorithm 7: 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  $\Gamma_g$ ;

    Evaluate  $\theta_i$ ,  $\forall \theta_i \in \Gamma_g^m$ ;

    Calculate  $b_i = \phi(\theta_i) = [\dots, E(o_{t_k}), \dots, E(o_{t_K})]$   $\forall \theta_i \in \Gamma_g^m$ ;

    Calculate  $\eta(\theta_i) = \frac{1}{|J|} \sum_{j \in J} \text{dist}(b_i, b_j)$ ,  $\forall \theta_i \in \Gamma_g^m$ ;

    Calculate  $s(\theta_i) = \sum_{k \in K} \|o_{t_k}^{(\theta_i)} - D(E(o_{t_k}^{(\theta_i)}))\|^2$   $\forall \theta_i \in \Gamma_g^m$ ;

$\mathcal{A}_{Nov} \leftarrow N_Q$  samples from  $\Gamma_g^m$ ;

**if**  $\phi(\theta_i) \in \mathcal{B}_{Rew}$  **then**

        |  $\mathcal{Q}_{Cand\_Em} \leftarrow \theta_i$

**end**

    /\* NSGA-II based policy selection \*/

    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** *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  $\theta_i \in F_J$ ;

**end**

**end**

---

the last observations  $o_T$  generated by the policies of the last  $I$  generations. On the contrary, **STAX** takes advantage of the alternating two-step process inherited from **SERENE**. During this process, the algorithm generates two collections of policies: the *reward archive*  $\mathcal{A}_{Rew}$  and the *novelty archive*  $\mathcal{A}_{Nov}$ . The dataset  $DS$  is then formed with the observations generated by the policies in both archives, plus the observations generated by the last population  $\Gamma_g$  and the last offspring population  $\Gamma_g^m$ . The data of the archives provides a *curriculum*, preventing the search to cycle back to already explored areas. This also prevents any possible destabilization of the training process due to the ever changing data distribution used in **TAXONS**. Another destabilizing factor can be the training of the **AE** on data collected thanks to the **AE** itself. This problem should be attenuated by training the model also on the data from policies in  $\mathcal{A}_{Rew}$ . These policies are in fact selected not according to the novelty calculated thanks to the **AE**, but only for their reward, that is a factor independent from the feature space. 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. Moreover, for each policy, the observations used to form the training dataset are all the observations used to generate the behavior

descriptor defined in Eq. (6.2). This is different from what done for TAXONS, where only the last observation  $o_T$  was used.

Once the dataset  $DS$  has been collected, it is split into two sub-datasets: *training dataset*  $DS_{\text{Train}}$  and a *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  $D_{\text{Val}}$ . The training episode is stopped if the error increases for 3 consecutive epochs. Contrary to TAXONS, here the training episodes do not happen at regular intervals. Instead, the AE is trained less and less frequently the longer the search is performed; this is the same strategy employed in the AURORA method [142]. Training according to this strategy allows to adapt the frequency of the training to the maturity of the learned BS. In fact, after the first training episodes, the learned representation is mature enough for the AE to start focusing on its refinement. This means that there is no reason anymore to train the AE too often, allowing to save time and computational resources. 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. The complete STAX algorithm is shown in Alg. 8.

### 6.2.3 Reward exploitation in a learned space

As it was the case for SERENE, STAX exploits any discovered reward through emitters. The power of this kind of approach has already been discussed in Chapter 5, where thanks to simple elitist reward-based emitters, SERENE could easily exploit any reward area discovered during the search. The ability to disjointly optimize multiple reward areas in an efficient way is even more fundamental for an approach like STAX. In hand-designed BS, like the ones SERENE was designed to deal with, the engineer has total control over the BS itself. This can help in reducing 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. It can happen that this space will have multiple reward areas, even if only one is present in the ground-truth BS. This can be explained by considering the Hard Maze environment with just one reward area shown in Fig. 6.1. In this example, the environment has a 2D ground-truth BS consisting of the  $(x, y)$  position of the robot at the end of a trajectory. On the contrary, the BS obtained by stacking the AE representations extracted from multiple RGB

---

**Algorithm 8: STAX**

---

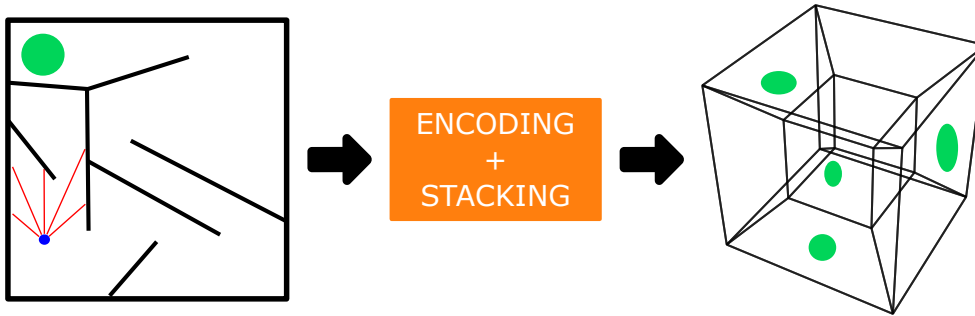
**INPUT:** evaluation budget  $Bud$ , budget chunk size  $K_{Bud}$ , population size  $M$ , emitter population size  $M_{\mathcal{E}}$ , offspring per policy  $m$ , mutation parameter  $\sigma$ , number of policies added to novelty archive  $Q$ , **AE** training interval  $TI$ , randomly initialized **AE**;  
**RESULT:** Novelty archive  $\mathcal{A}_{Nov}$ , rewarding archive  $\mathcal{A}_{Rew}$ , trained **AE**;

$\mathcal{A}_{Nov} = \emptyset$ ;  
 $\mathcal{A}_{Rew} = \emptyset$ ;  
 $\mathcal{Q}_{Em} = \emptyset$ ;  
 $\mathcal{Q}_{Cand\_Nov} = \emptyset$ ;  
 $\mathcal{Q}_{Cand\_Em} = \emptyset$ ;  
 $D = 0$ ;

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  $b_i = \phi(\theta_i) \in \mathcal{B}, \forall \theta_i \in \Gamma_0$ ;  
    **end**  
    Exploration Phase ( $K_{Bud}, m, \sigma, \mathcal{A}_{Nov}, \mathcal{Q}_{Cand\_Em}, \Gamma_g, Q, \mathbf{AE}$ );  
     $TI_C = TI_C + 1$ ;  
    **if**  $TI_C == TI$  **then**  
         $DS = \text{Extract dataset}(\mathcal{A}_{Nov}, \mathcal{A}_{Rew}, \Gamma_g, \Gamma_g^m)$ ;  
        Train Autoencoder (**AE**,  $DS$ );  
        Update descriptors (**AE**,  $\Gamma_g, \Gamma_g^m, \mathcal{A}_{Nov}, \mathcal{A}_{Rew}, \mathcal{Q}_{Em}, \mathcal{Q}_{Cand\_Nov}, \mathcal{Q}_{Cand\_Em}$ );  
         $TI = TI + 1$ ;  
         $TI_C = 0$ ;  
    **end**  
    **if**  $\mathcal{Q}_{Cand\_Em} \neq \emptyset$  **or**  $\mathcal{Q}_{Em} \neq \emptyset$  **then**  
        Exploitation Phase ( $K_{Bud}, \mathcal{Q}_{Cand\_Em}, \lambda, m, \mathcal{Q}_{Em}, \mathcal{A}_{Nov}, \mathcal{A}_{Rew}, M_{\mathcal{E}}$ );  
    **end**  
**end**

---

high-dimensional observations has higher dimensionality. This can lead to multiple areas of this higher dimensional space representing the ground-truth reward area. 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. This approach can be even more



**Figure 6.1:** *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.*

useful in settings in which the reward is difficult to express in the ground-truth BS but easy to observe from the high-dimensional observations of the trajectory. Being able to see the rewarding situations from these observations means that it will be likely for them to be represented in the learned BS. This will create zones in this space in which the reward can be achieved, allowing the algorithm to use emitters to exploit it. The whole exploitation phase is shown in Alg. 9.

### 6.3 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 task at hand and the environment. The performances of STAX will be compared against various baselines to study which ones are the most important aspects of the method.

In order to perform this analysis STAX is evaluated on 3 of the sparse rewards environments presented in Chapter 5:

**Curling:** it consists of a 2 DoF arm pushing a ball over a table [161]. The arm is controller 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 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  $(x, y)$  position of the ball. The environment is shown in Fig. 6.2. Fig. 6.2.(b) shows the  $64 \times 64$  RGB image the AE sees during the algorithm execution.

**HardMaze:** it consists of a 2-wheeled robot whose goal is to navigate a maze with the aid of 5 distance sensors [34]. The robot, in blue in Fig. 6.3, 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. 6.3.(a),

---

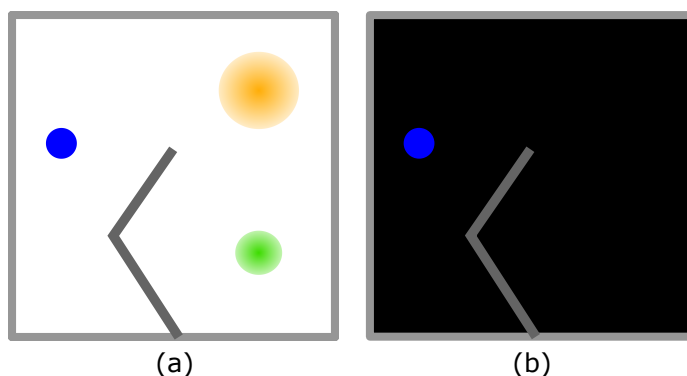
**Algorithm 9: STAX** Exploitation Phase

---

**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  $\mathcal{A}_{Nov}$ ;

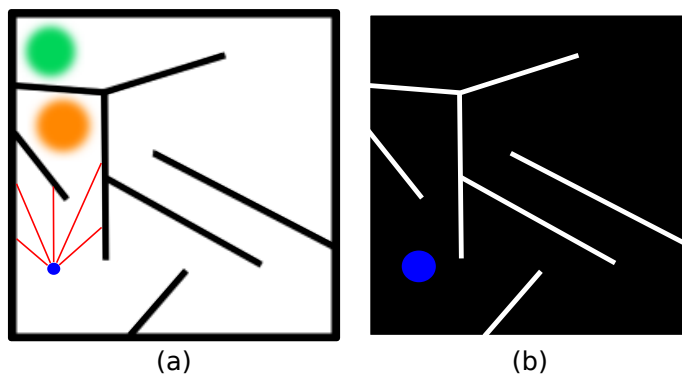
```
/* Bootstrap step */
while  $K_{Bud}/3$  not depleted do
    Select most novel policy  $\theta_i$  from  $\mathcal{Q}_{Cand\_Em}$ ;
    Calculate  $\sigma_i$ ;
    Initialize:  $\mathcal{E}_i$ ,  $\mathcal{Q}_{Cand\_Nov}^i = \emptyset$ , and  $P_0$ ;
    for  $\gamma \in \{0, \dots, \lambda\}$  do
        if  $P_0$  then
            Evaluate  $\tilde{\theta}_j, \forall \tilde{\theta}_j \in P_0$ ;
        end
        Generate offspring population  $P_\gamma^m$  from  $P_\gamma$ ;
        Evaluate  $\tilde{\theta}_j, \forall \tilde{\theta}_j \in P_\gamma^m$ ;
        Generate  $P_{\gamma+1}$  from best  $\tilde{\theta}_j \in P_\gamma^m \cup P_\gamma$ ;
    end
    Calculate  $I(\mathcal{E}_i)$ ;
    if  $I(\mathcal{E}_i) > 0$  then
         $\mathcal{Q}_{Em} \leftarrow \mathcal{E}_i$ ;
    end
end
/* Emitters step */
Calculate pareto fronts in  $\mathcal{Q}_{Em}$ ;
while  $2/3K_{Bud}$  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  $P_\gamma$ ;
        Evaluate  $\tilde{\theta}_j, \forall \tilde{\theta}_j \in P_\gamma^m$ ;
         $\mathcal{A}_{Rew} \leftarrow \tilde{\theta}_j, \forall \tilde{\theta}_j \in P_\gamma^m \mid r(\tilde{\theta}_j) > R_\gamma$ ;
         $\mathcal{Q}_{Cand\_Nov}^i \leftarrow \tilde{\theta}_j, \forall \tilde{\theta}_j \in P_\gamma^m \mid \eta(\tilde{\theta}_j) > \eta_i$ ;
        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
             $\mathcal{A}_{Nov} \leftarrow N_Q$  samples from  $\mathcal{Q}_{Cand\_Nov}^i$ ;
            Discard emitter  $\mathcal{E}_i$ ;
        end
    end
end
end
```

---



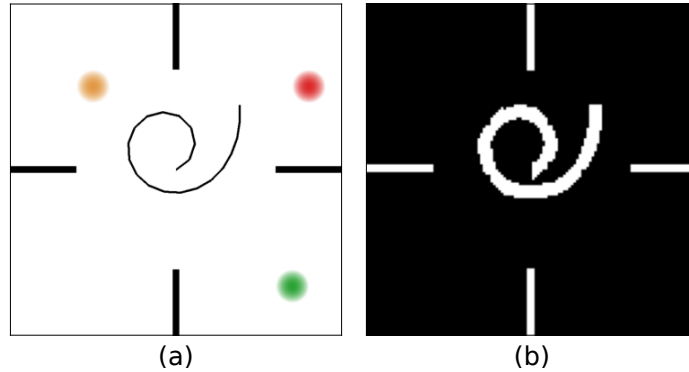
**Figure 6.2:** *Curling environment. (a) Real environment with reward areas. (b) RGB observation of the environment as seen by the AE.*

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  $(x, y)$  position of the robot. The whole environment is shown in Fig. 6.3.(a), while in Fig. 6.3.(b) is shown the  $64 \times 64$  RGB observation fed to the AE.



**Figure 6.3:** *HardMaze environment. (a) Real environment with reward areas. (b) RGB observation of the environment as seen by the AE.*

**Redundant Arm:** it consists of a 20-DoF arm moving on a 2 dimensional plane [164]. 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  $(x, y)$  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 is shown in Fig. 6.4.(a). Fig. 6.4.(b) shows the  $64 \times 64$  RGB image that the AE uses to build the behavior descriptors.



**Figure 6.4:** *Redundant Arm environment. (a) Real environment with the 3 reward areas. (b) RGB observation of the environment as seen by the AE.*

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

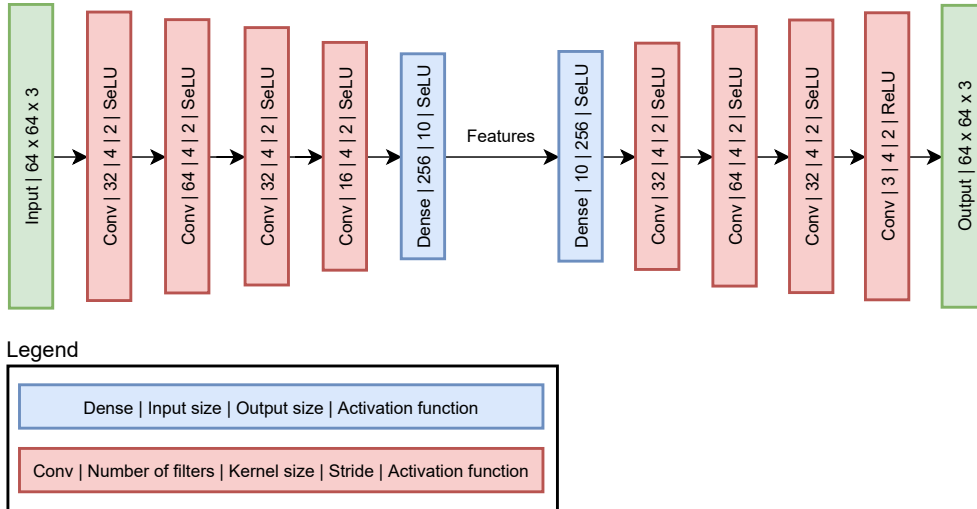
[STAX](#) is compared against the following baselines:

- [NS](#) [34]: vanilla [NS](#), that performs pure exploration and does not attempt to improve on the reward;
- [ME](#) [43]: vanilla MAP-Elites that uses a  $50 \times 50$  grid to cover the behavior space of every environment;
- [MOO-NR](#) [115]: the multi-objective evolutionary algorithm introduced in Sec. 2.2.1. It optimizes both the novelty and the reward of the policies;
- [TAXONS](#) [48]: it performs pure exploration by learning the behavior descriptor through an [AE](#) trained during the search process;
- [SERENE](#) [161]: it performs exploration through [NS](#), 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 used 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 for it, shown in Fig. 6.5. 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 [148], 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 [149] with an learning rate of 0.001. Finally, the statistical results are computed over 15 runs



**Figure 6.5:** AE structure. The input and output of the network are represented in green; in red are the convolutional layers, while in blue are the fully connected layers.

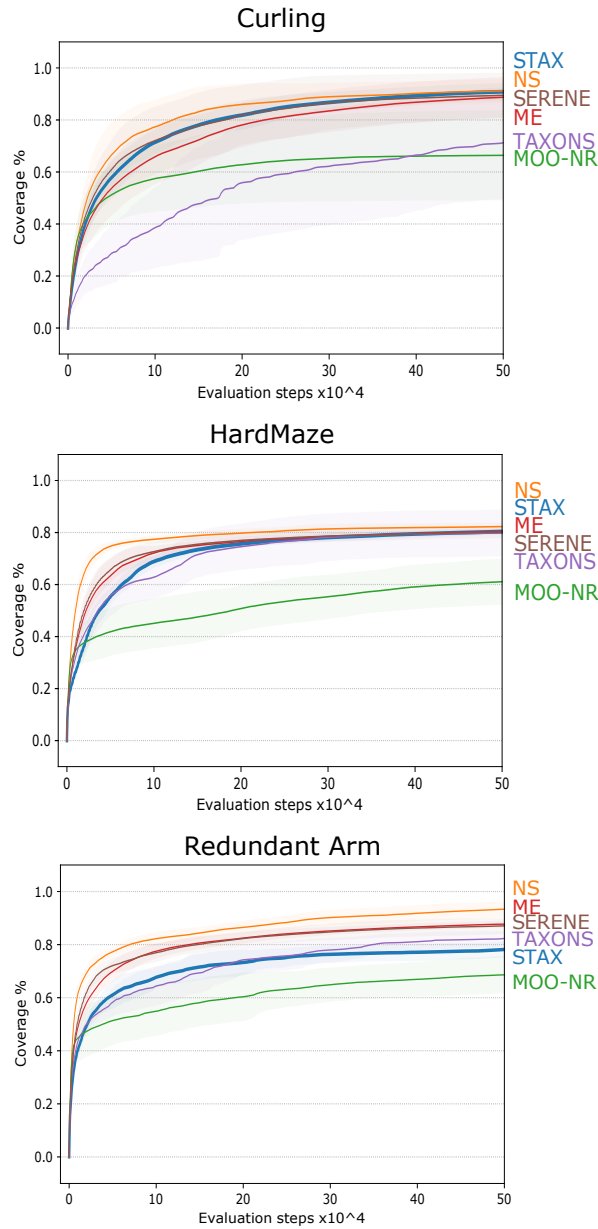
for each experiment.

## 6.4 Results

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

### 6.4.1 Exploration

Performing good exploration in situations of sparse rewards is fundamental but is not an easy task, even more so with a minimal amount of prior information, as STAX does. This section studies how well STAX can explore. This is done by measuring the *coverage metric* obtained in the *ground truth BS* that are defined in Sec. 6.3 for each one of the tested environments. As done in previous chapters, the coverage metric is calculated 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



**Figure 6.6:** Average coverage with respect to the given evaluation budget reached by *STAX* against the different baselines. The shaded areas represent one standard deviation.

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 in areas of the learned space that are not considered by the coverage metric in the ground-truth space.

Fig. 6.6 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. All of this while having minimal information about the environment and the task. At the same time, the methods using the hand-designed ground-truth [BS](#) to drive the search, that is [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. This is not the case for [MOO-NR](#), which struggles in all environments, likely because of the interference of the multi-objective optimization between reward and diversity maximization.

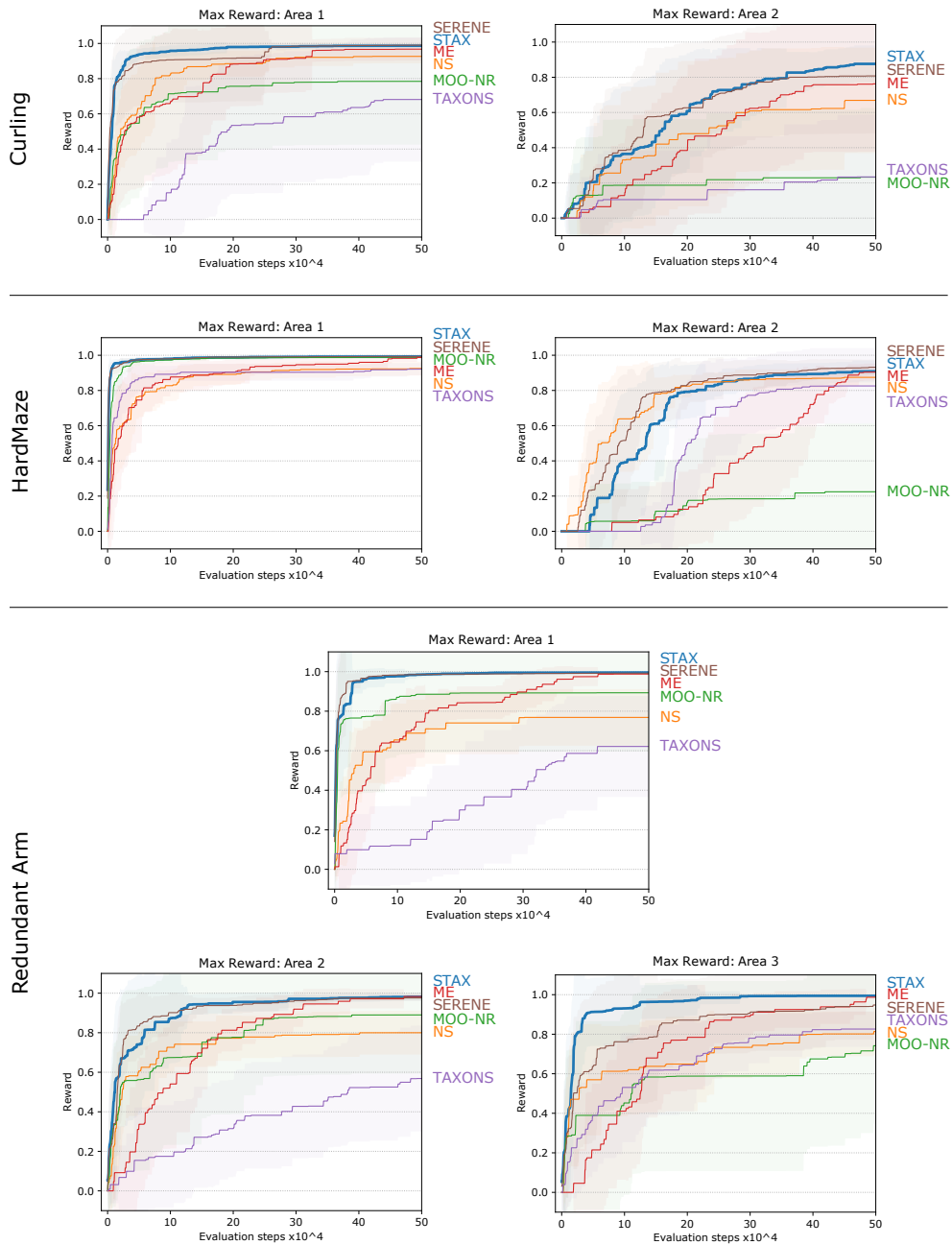
[TAXONS](#) also obtains high coverage, with the notable exception of the Curling environment. This is in contrast with the results obtained in Chapter 3, where the coverage of [TAXONS](#) in the Billiard environment was comparable to the one of [NS](#). The culprit of this loss of performance is likely the presence of the 2-Dof arm in the image fed to the [AE](#), as shown in Fig. 6.2. The arm can act as a distractor and was not present in the experiments done in Chapter 3. 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 more efficient selection of new policies according to the pareto based approach, performed by [STAX](#), and the training of the [AE](#) on the observations from policies in the reward archive. This last element allows [STAX](#) to observe many more situations in which the ball is in different positions.

### 6.4.2 Exploitation

Fig. 6.7 shows the maximum reward achieved by the algorithms in all the reward areas. Using emitters to exploit the reward allows [STAX](#) to reach almost the maximum reward in very few evaluations. These performances are very 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, which suffers from the same problems highlighted in Chapter 5, maximizing only the easiest to reach reward area. 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. 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.

### 6.4.3 Final archives distribution

Fig. 6.8 shows the final distribution of the behaviors representations for the policies in the final archives. Each point represents a different 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



**Figure 6.7:** Average maximum reward reached in all the reward areas by *STAX* against the different baselines. The shaded areas represent one standard deviation.

any reward, thus 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 as said in Chapter 5, 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 its discretization.

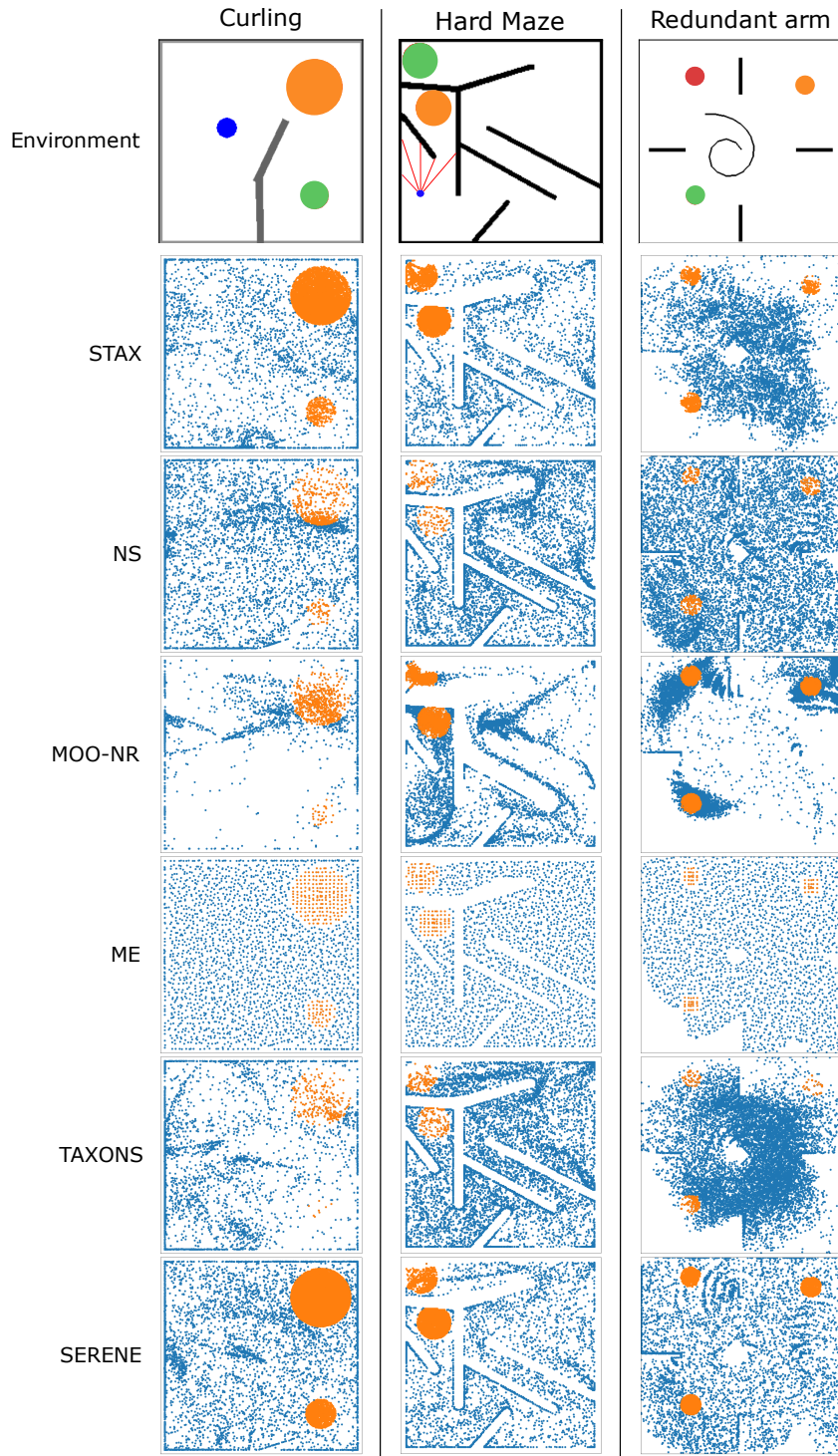
#### 6.4.4 Exploration ablation studies

This section studies what are 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.

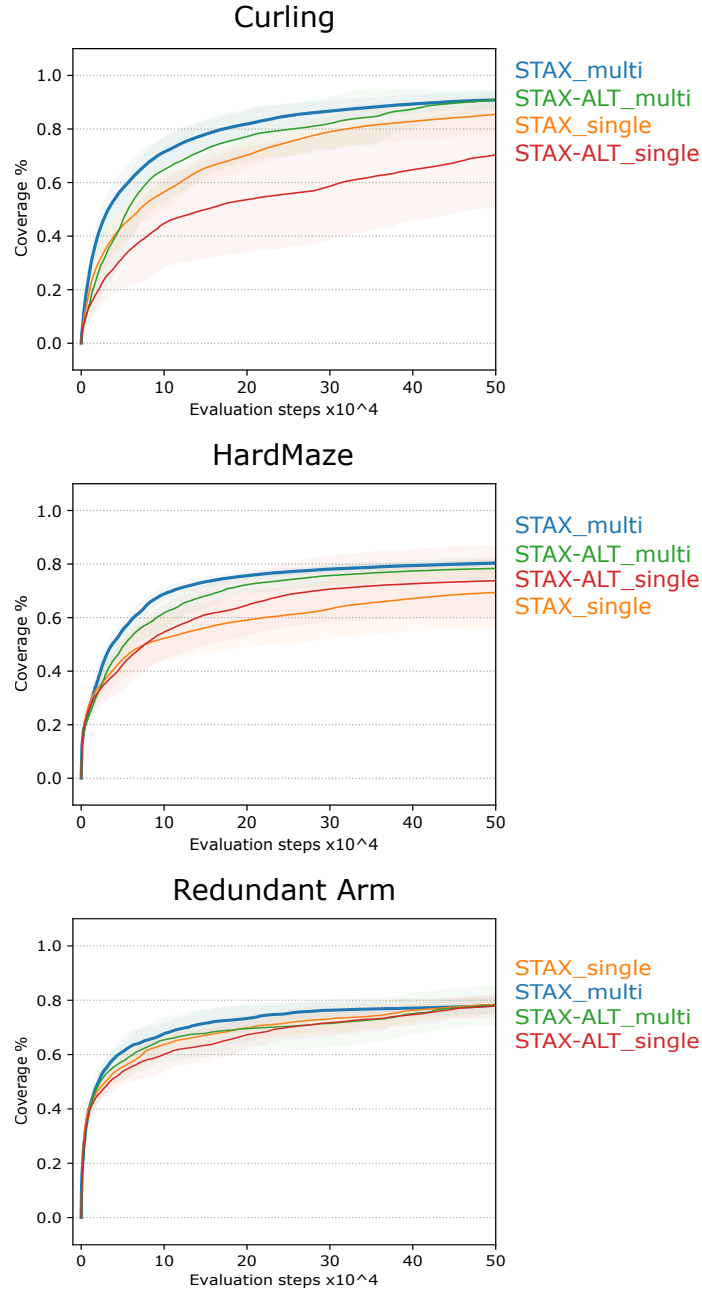
Both the coverage metric, calculated as in previous sections, and the maximum reward reached by each variant over each reward areas are analyzed. It can seem counterintuitive to analyze the maximum reward obtained by algorithms separating exploration from exploitation when performing an ablation study on the factors fostering exploration. The importance of this analysis lies in the fact that **STAX** and its variants select emitters also according to the novelty of the original policy. This means that the way the novelty is calculated has an influence - albeit minimal - on the reward exploitation. Moreover, being in a setting of sparse rewards, the way exploration is performed has a direct influence on the speed each reward area is discovered and optimized.

The results on the average coverage reached by the algorithms are shown in Fig. 6.9. From it, it is possible to see that the variants using multiple observations of the trajectory tend to 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



**Figure 6.8:** *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.*

observations tend to reach higher levels quicker. A possible explanation for 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.

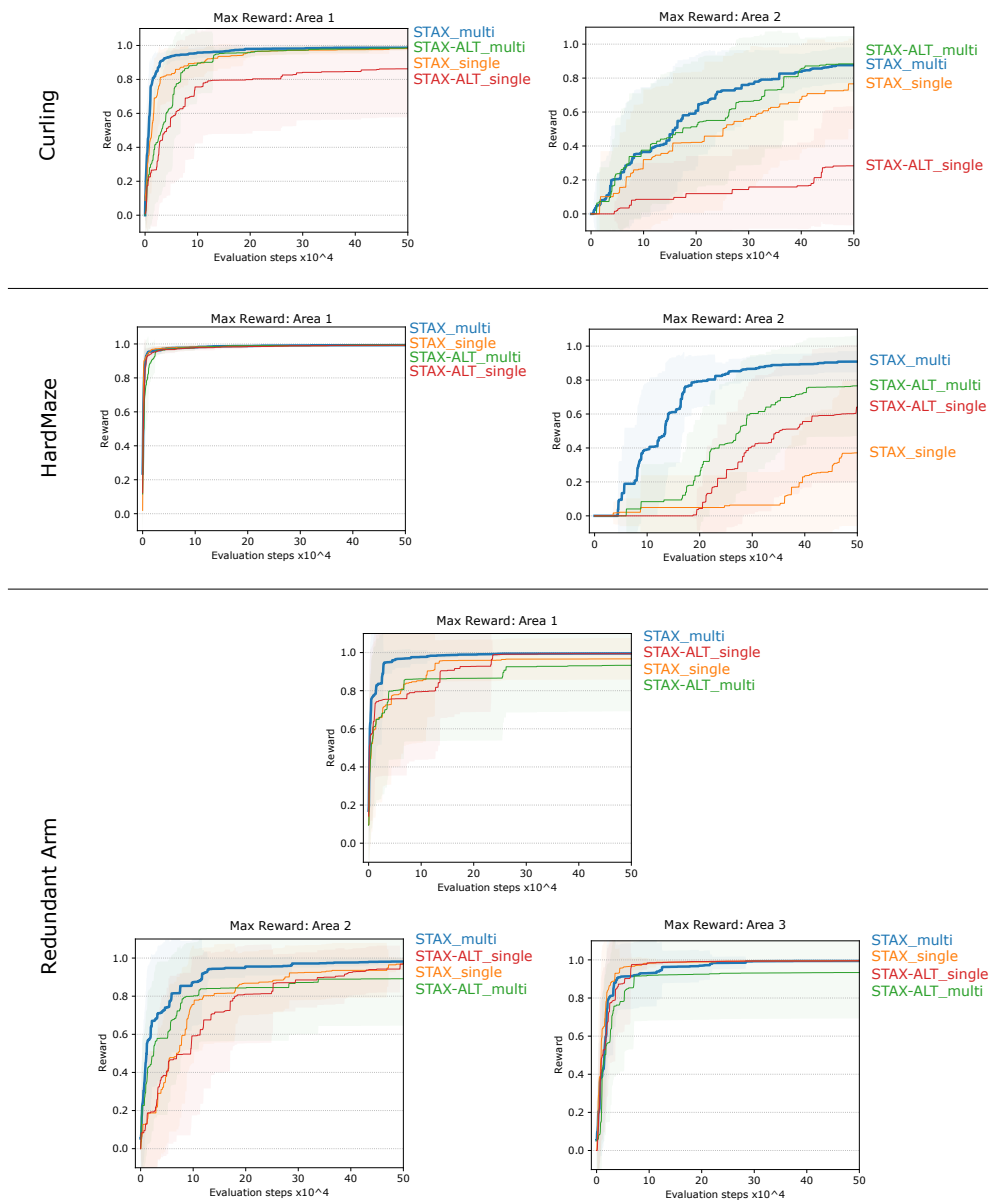


**Figure 6.9:** Average coverage with respect to the given evaluation budget reached by *STAX* against the ablated versions of the algorithm. The shaded areas represent one standard deviation.

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. 6.10. 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 performs 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.



**Figure 6.10:** 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.



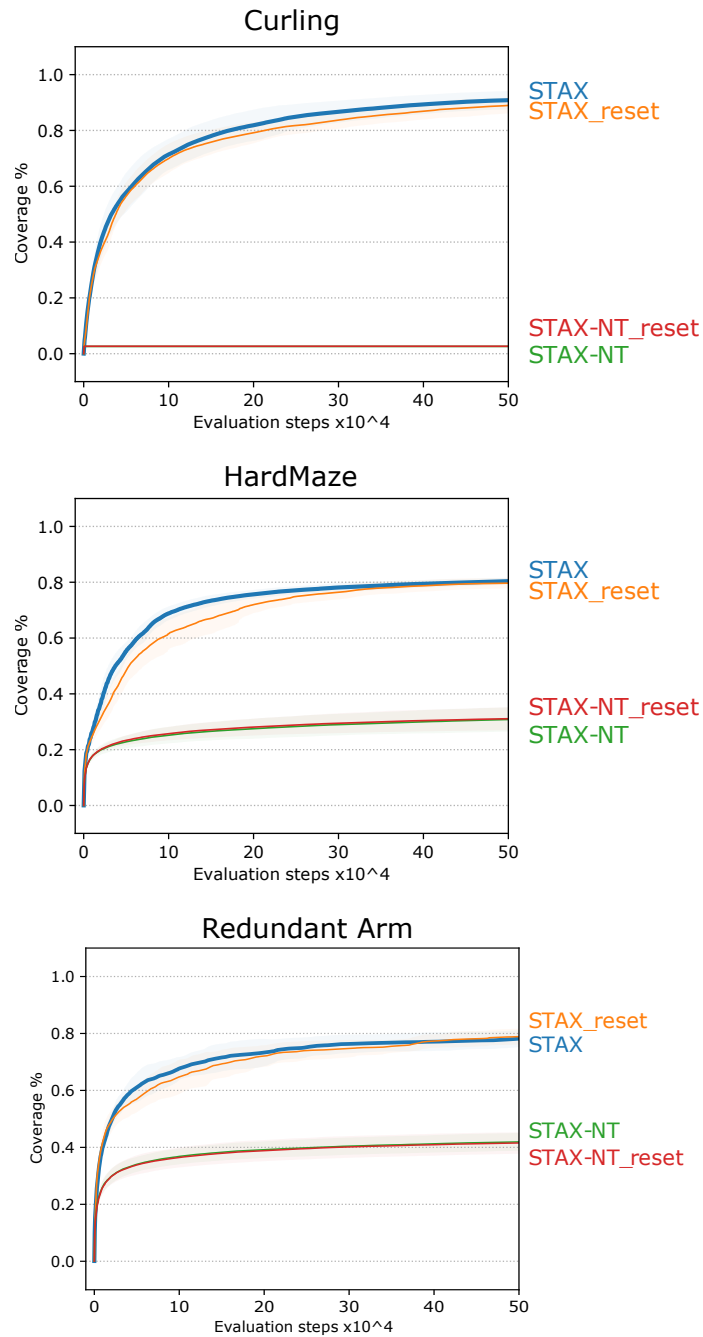
### 6.4.5 Autoencoder training regime

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. The first one concerns how important it is to learn the representation versus just using a random one. The second aspect is the importance of continuously training the **AE** during the whole search process. This training strategy 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. However, during the first iterations of the search, when the archives are still small, the **AE** will mainly be trained on the data coming from the last population and offsprings. This means that at the beginning of the search the biggest contribution to the "memory" of already explored areas for the **AE** comes from the continuous training of the **AE**.

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

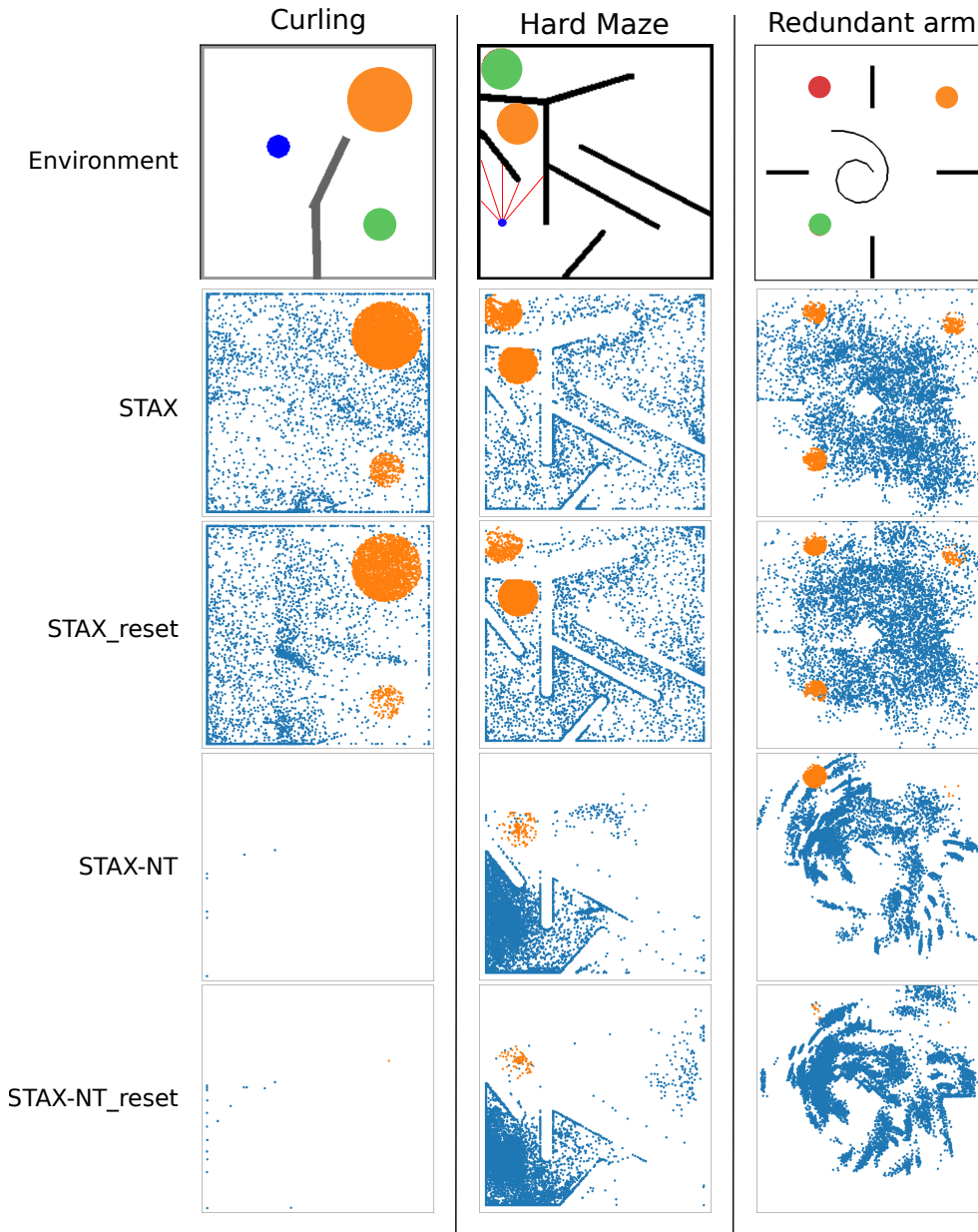
- **STAX-NT**: in which the search is driven through an **AE** whose weights are randomly sampled at the beginning of the search and not modified anymore;
- **STAX-NT\_reset**: in which the search is driven through an **AE** whose weights are randomly sampled every  $TI$  exploration steps. This means 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 version of **STAX** is the **AE** training regime. The behavior descriptor is still generated by stacking the representations extracted from 5 frames sampled along the trajectory, as done in Sec. 6.4.1. The coverage results for the 3 tested environments are shown in Fig. 6.11. Not surprisingly, the results show that training the **AE** rather than using a randomly generated one really pushes exploration. This means that 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 results show how 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, **STAX** has a much smaller execution wall time compared to **STAX\_reset**, not having to retrain the **AE** from scratch every time.



**Figure 6.11:** 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.

Fig. 6.12 shows the final distribution of the archived policies behavior descriptors. 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, in order to obtain good ex-



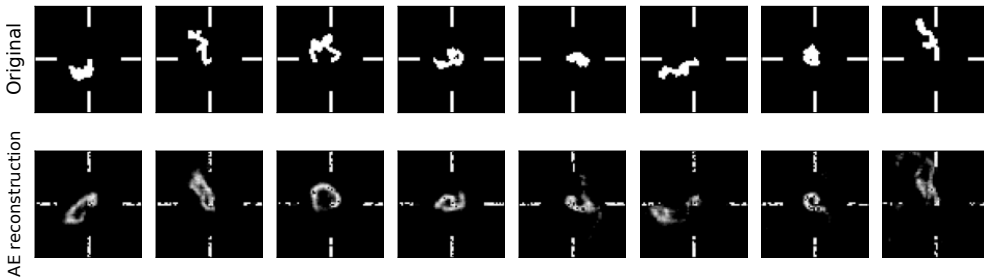
**Figure 6.12:** *Distribution of the behavior descriptors of the archived policies. On each column are shown the results for an environment, while on each row is shown the distribution for each experiment. The archive plotted are from the runs achieving highest coverage. In blue are the policies in the novelty archive and in orange the policies in the reward archive.*

ploration 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 farther.

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.

### 6.4.6 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. 6.13 shows how well STAX’s learned AE can reconstruct the observations collected during the evaluation of the policies. The first row shows the  $64 \times 64 \times 3$  final observation of the trajectories of a set of policies sampled from the final archives. In the second 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. 6.4.1.



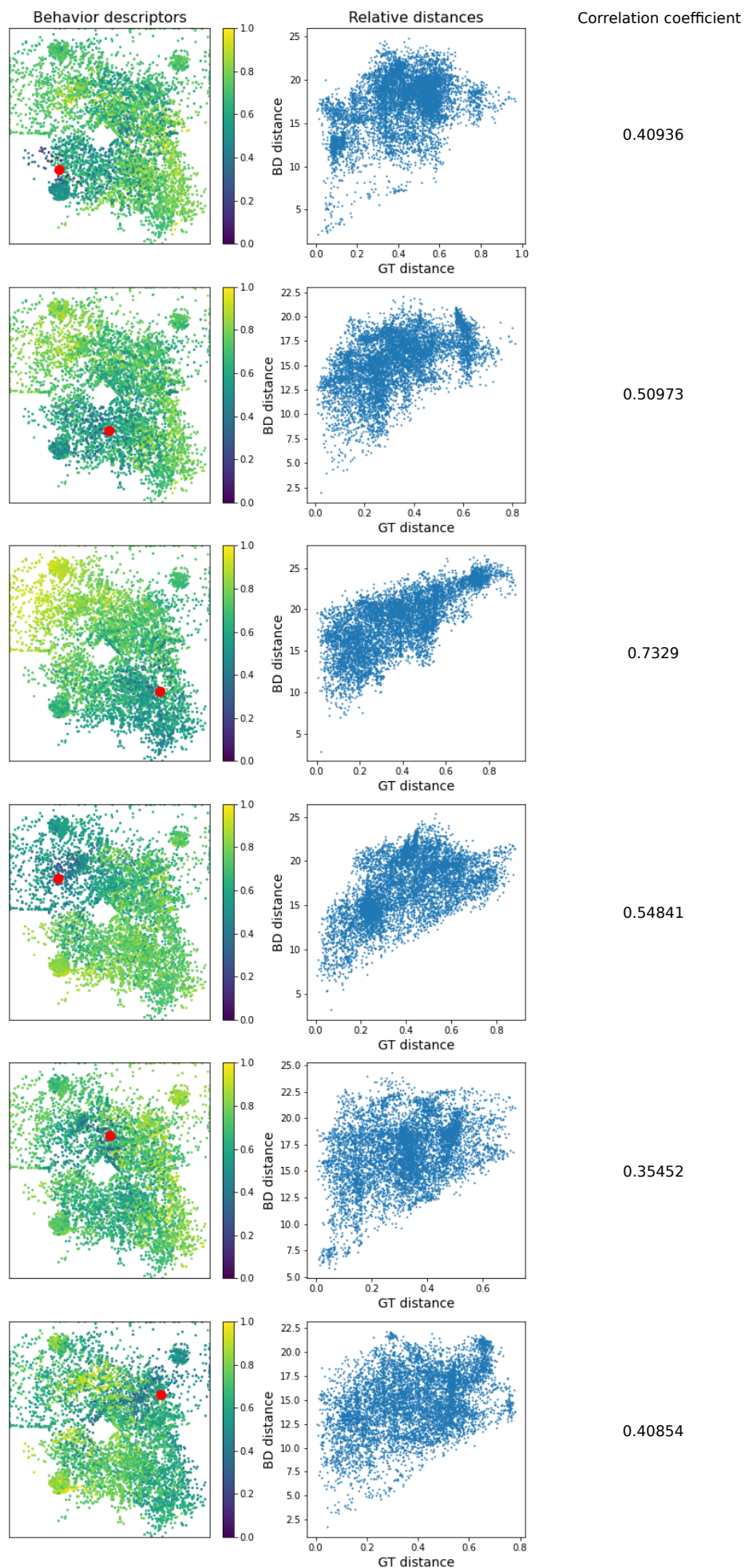
**Figure 6.13:** Reconstruction of the AE trained during the search performed by STAX. The first row shows the original  $64 \times 64 \times 3$  images. The second row shows the reconstructions of the images produced by the AE.

From this, 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. 6.14. 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 [167] between these distances.

From the figure it is possible to see that closer points in the learned **BS** are closer in the ground-truth space, and the farther this 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. The correlation coefficient shows 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.



**Figure 6.14:** 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.



## 6.5 Conclusion

This chapter introduced **STAX**, a method that reduces the amount of prior information about a task to minimal levels. It does so by combining the representation learning ability of **TAXONS** when dealing with unknown **BS** and the capacity to focus on interesting areas of the search space of **SERENE** 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 limitation of dealing with environments where the final observation needs 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. 6.2.3, performing exploitation through emitters that can deal with multiple disjoint reward areas when learning a **BS** representation can prove extremely useful. This is due to the fact that there is no guarantee that the learned **BS** will represent all the rewards in a single connected area.

**STAX** has been tested on three different sparse rewards environments, reaching high performances in all of them, both from the point of view of exploration and exploitation of the reward. These results are comparable to the ones obtained by **SERENE** notwithstanding **STAX** being provided much less prior information about the task to solve. Moreover, thanks to the representation learning capability provided by **TAXONS**, **STAX** is capable of overcoming the main limitation of **SERENE** discussed in Sec. 5.6.

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, the Chapter shows how the learned **BS** has a similar structure to the ground truth **BS**, allowing the algorithm to perform good exploration in both.

Notwithstanding the multiple shortcomings of the original **NS** algorithm addressed by **STAX**, there are still many aspects of this method that can be further studied and improved. 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 stimulating direction of research is the one initiated by Cully in [56], where multiple kind of emitters are combined through a multi-armed bandit approach.





**Chapter content**


---

<b>7.1 Learning the behavior space . . . . .</b>	<b>108</b>
7.1.1 Distractors . . . . .	110
7.1.2 Disentangled representations . . . . .	111
<b>7.2 Focusing on the interesting parts of the search space</b>	<b>111</b>
<b>7.3 Noisy environments . . . . .</b>	<b>112</b>

---

This manuscript introduced methods to augment the capacity of [NS](#), and [QD](#) algorithms in general, when dealing with the problem of sparse rewards. The main approach considered is to focus on exploration while reducing the amount of prior information needed by the algorithm. The contributions can be grouped in two main categories:

- Autonomously learning the behavior space while performing the search;
- Giving the methods the ability to focus on the most interesting areas of the search space without sacrificing global exploration.

This chapter discusses these contributions in light of the introduced methods. Their limitations are also analysed, with possible solutions and extensions, and the research directions they open.

**7.1 Learning the behavior space**

The biggest contribution towards autonomously learning the search space is the one provided by [TAXONS](#), discussed in [Chapter 3](#). By learning the space in which the novelty of the policies is evaluated, this method greatly reduces the amount of engineering effort needed at design time. At the same time, it extends the range of domains in which [NS](#) and [QD](#) algorithms can be applied. For many problems it is not easy to properly hand-design a [BS](#) that can foster good exploration. For example, while trying to discover policies capable of changing the position of an object, this position needs to be tracked. This usually implies having multiple cameras capable of triangulating the location of said object, or a dedicated software. Learning the behavior space through high-dimensional RGB images removes this need, at the cost of not being able to control the representation of the [BS](#) anymore. [Sec. 3.4.3](#) showed how well

**TAXONS** can perform exploration by observing the environment through said RGB images.

Most of the prior information provided to **TAXONS** comes from the selection of the final observation of the trajectory to generate the behavior descriptor. This requires the final state of the whole system to contain information about the whole behavior of the policy. A study in order to address this limitation was conducted in Chapter 4. The focus of the study was the signature transform [54], a mathematical operator capable of encoding a whole stream of data into a single vector. This could have been combined with **TAXONS** through a two step process. First, the **AE** would represent each high-dimensional RGB image generated during the trajectory in its learned feature space, generating a trajectory of encoded representations. Then the signature could be used on this trajectory to encode it even more, in a single, fixed length vector. Before continuing on this path, the signature was tested as a way to encode a trajectory of the simple ground-truth representations generated during the policy evaluation. The results reported in Sec. 4.5 show that even if signatures can foster good coverage along the whole trajectory of states, simply sampling few observations from the trajectory allows to obtain similarly good results. The probable reason is the exponential increase in dimensionality of the signature with respect to its order and to the size of the datapoints. It has been shown that in high-dimensional spaces, distance metrics lose most of its relevance [143]. This can hinder the exploration power of **NS** based algorithms that rely on distances in the **BS** to push for diversity. For this reason, when **STAX** was introduced in Chapter 6, the method relied on a subsampling and stacking of multiple observations along the trajectory to generate the **BS**. Even if relatively simple, this approach proved effective enough in greatly reducing the amount of prior information with minimal overhead. Sec. 6.4 shows that thanks to this approach, **STAX** can autonomously learn a behavior descriptor from the whole trajectory, fostering good exploration in the unknown ground-truth **BS**.

However, autonomously learning a representation of the **BS** does not come without costs. The training of the **AE** adds significant overhead in both execution time and computation power required to execute the algorithm. Using an **AE**, and even more so one based on **Convolutional Neural Network (CNN)**, requires the presence of a dedicated GPU to speed up training and inference, requirement not usually necessary for **QD** algorithms with an hand-crafted **BS**. Moreover, after every training episode of the **AE**, the descriptors of the policies in all archives and current populations need to be updated with the features extracted by the newly trained **AE**. This makes the methods scale badly with the number of generations, given the continuously increasing number of policies in the archives that need to be updated. The problem is much more important for **TAXONS** than for **STAX**: the former trains the **AE** at regular intervals while the latter performs the training episodes less frequently with the passing of generations.

At the same time, these are not the biggest shortcomings. As said, by autonomously learning the **BS** the designer loses control on what features will

be present in the descriptors. The **AE** will try to learn a representation of everything present in the RGB observations. This includes distractors.

### 7.1.1 Distractors

Stone et al. define distractors as “variations in the input that are irrelevant for the task” [153]. They can appear in multiple forms. It is possible to have distractors controlled by the algorithm but not directly related to the task, or distractors whose presence or actions are independent from the learning algorithm. An example of the first kind of distractors is the arm in the Curling environment used in Chapter 6 to test **STAX**. While directly controlled by the policies, the position of the arm at any given moment is not important with respect to the task of having the ball in different locations on the table. At the same time, being represented in the RGB observations, it can greatly influence the novelty of the corresponding behavior descriptor extracted from these observations. In the case of the Curling experiments, said distractor did not prove to be a limitation for **STAX**, as it can be seen from the results reported in Chapter 6. However, this is more likely due to the simplicity of the environment rather than the power of the algorithm itself. Nonetheless, learning to represent this kind of distractors can be beneficial to the generalization capabilities of the algorithm. The final archive, selected according to diversity in a space in which the distractor is encoded, can then be used to address tasks for which what before was a distractor now is fundamental.

The second kind of distractors, the ones on which the algorithm has no control, can be more difficult to deal with. These can be of different types, going from an object randomly moving in the environment to the background changing color. These aspects of the environment can be problematic for any algorithm using intrinsic rewards to foster exploration. This is due to the great source of randomness provided by these elements, rendering already visited states novel enough for the algorithm to keep revisiting them. The problem has been formalized as a thought experiment and given the name of *Noisy TV problem* [87]. The experiment considers an agent in an environment trying to maximize the novelty of its observed states. The environment contains a TV in which random images are continuously displayed. This TV would be able to attract the focus of the agent forever, thanks to the continuous novelty provided by the unpredictability of these random images. In these situations, the agent would need an external signal in order to understand what is interesting and what is not. This is especially true when the representation is learned in an unsupervised fashion as done for **TAXONS** and **STAX**. The importance of the problem of distractors and the increased attention to it are highlighted by the amount of recent publications proposing methods aiming at dealing with distractors [98, 87, 168, 169, 170, 171, 172]. Moreover, Stone et al.[153] extended the classical DeepMind control suite [173] with visual distractors in order to benchmark existing algorithms against this kind of problem.

While the methods introduced in this manuscript cannot deal with them,

studying a way to address distractors in the framework of QD algorithms can be an exciting line of future work. This could also lead to a much wider range of application of this family of algorithms.

### 7.1.2 Disentangled representations

An interesting approach in dealing with distractors is to learn a set of disentangled representations of the BS [174, 175]. Disentangled representation learning allows to separate, or *disentangle*, each independent factor of variation in the input vector as a small set of variables in the learned feature space. In other terms, it allows to separate the factor of variations in the inputs on different axis in the feature space. Being able to do so would allow the designer to easily select on which aspects of the learned BS the algorithm should focus, thus also being able to steer the search in the desired direction.

There are many ways in which this problem can be tackled, all falling under the umbrella of state representation learning. This field is rapidly growing and attracting the interest of many researchers [137]. At the same time, there are many aspects to consider. It was shown that through pure unsupervised learning it is impossible to obtain a completely disentangled representation of the factors of variation [176]. This means that some other factors need to be taken in account while trying to learn this type of representations. In this regard, an interesting approach is HOLMES [177], in which the authors learn a hierarchy of representations to foster exploration. While the approach is designed for morphogenetic systems, it should be possible to apply it to settings similar to the ones addressed in this manuscript. A related problem is that the autonomous learning of a *single representation space*, in which exploration is performed, can be limiting on more complex problems, where multiple aspects may need to be explored. The collection of policies can then be biased toward a single one of these aspects. MC-AURORA [178] addresses this problem by generating multiple collections. For each of these collections, the policies are selected according to a different learned representation. This helps in having a more diverse final collection of policies, covering many of the aspects of a same problem. Generating multiple archives thanks to multiple learned representations can be an interesting direction of research also for methods like TAXONS and STAX.

## 7.2 Focusing on the interesting parts of the search space

NS's limitation of not focusing on any interesting part of the search space was addressed in this manuscript in Chapters 5 and 6. The main contribution is the one introduced by SERENE, which can exploit any possible reward discovered during the search by using emitters. This idea has then been extended with STAX, a method in which the exploration is driven not by NS but by TAXONS, while still exploiting the reward through emitters. This way of exploiting the reward areas has proven to be effective in quickly obtaining

high rewards on all the disjoint reward areas, as shown in Sec. 5.5 and in Sec. 6.4. An important aspect of these two methods, greatly contributing to such results is the separation of the exploration and exploitation steps in two alternating processes. The switch between the two is performed thanks to a scheduler assigning computation budget to either one of them. The scheduler used in this manuscript is fairly simple: if any reward is discovered, it assigns the budget to the two processes in an alternating fashion, otherwise it completely focuses on exploration. Sec. 5.5.1 analyzed how this simple strategy is effective in balancing the budget between exploration and exploitation of the different reward areas. Nonetheless, there is still room for improvement. Once all the discovered rewarding areas have been densely explored and optimized upon, it would be best to focus more on the exploration in order to discover more reward areas. At the same time, after much of the search space has been explored, there is no need anymore to spend half the budget on exploration, but rather focusing on improving on the rewards could prove to be more beneficial. For this reason, having a scheduling strategy capable of adapting itself to the dynamics of the search process could help in optimizing the efficiency of the search even more. This issue is what is usually referred as *exploration-exploitation trade-off* [13]. A possible way to optimize the scheduler strategy in these situations could be to use a MAB approach [63], capable of selecting the best option among the two at each given moment. At the same time, to apply a MAB algorithm there is the need to define a metric to optimize between exploration and exploitation. MABs can in fact be described as stateless RL algorithms. This means that they learn a policy by optimizing a reward function. In order to define a MAB reward function for an emitter based approach like SERENE or STAX, there is the need for a way to measure the exploration progress and the reward exploitation in comparable ways. Even if this looks like an easy to address issue, it is not the case due to the fundamentally different nature of the two aspects.

Another possible direction of development for SERENE based algorithms, regards the type of emitters used. In this manuscript only a single type of elitist-based algorithm has been used as emitter, but this needs not to be the case. Different kind of emitters were already introduced in Fontaine’s work [55], even if a single type was used at the time. Cully improved on this by using multiple kind of emitters concurrently, selecting the best one at each given time through a MAB [56]. Moreover, emitters do not have to be necessarily based on EAs. On the contrary, any reward based algorithm can be used as an emitter, RL ones included. Following this path of research could help in improving the efficiency of emitter based algorithms and allow their application to more complex problems.

### 7.3 Noisy environments

Another limitation of NS based algorithms, and more in general all QD ones, is the inability to deal with noisy environments. This is due to the way the behavior representation of each policy is compared against all the others. In

order to have a meaningful comparison, the environment in which the policies act needs to be deterministic, having as only reason for the difference among behaviors the policy itself. The requirement for deterministic environments greatly hinders the range of problems to which this kind of algorithms can be applied. Many real world environments are in fact noisy by nature. For this reason, being able to deal with noisy environments is an important aspect to take into account, even more in comparison with the [RL](#) literature, for which stochastic environments are common. Yet, while some work has been done in dealing with noisy fitness functions for standard [EAs](#) [179, 180], not so much effort has been done with respect to [QD](#) algorithms. The most straightforward approach to the problem is to perform multiple evaluations of the same policy to strengthen its performance estimation when calculating its behavior descriptor [141]. The same idea has also been used in conjunction with applying adaptive sampling to increase the efficiency of the evaluation [130]. Yet, these approaches come with the added cost of testing multiple times the same policies, greatly increasing the computational requirements of the algorithm.

A different strategy is the one introduced with [DG-MAP-Elites](#) [131], in which the usual [ME](#) grid is extended in depth in order to host multiple policies in the same cell. This allows a better estimate of the performance of a given elite of solutions without the need for multiple evaluations. Said estimate leads to a higher stability of the archive with respect to noise. At the same time, while this strategy works for a grid based algorithm as [ME](#), it might be difficult to adapt to [NS](#) based algorithms. In this light, a different way to approach the problem could be to use a multi-objective approach in which policies are selected not only for their novelty but also for their robustness to noise. While this would still require multiple evaluations for each policy, it could lead to more robust and generalizable solutions.

Nonetheless, the problem is still far from solved, and working in this direction could lead to great improvements in the range of applicability and recognition from other communities of [QD](#) algorithms.



---

# Conclusion

Throughout this manuscript, algorithms capable of addressing the issue of sparse rewards with minimal prior information about the task have been introduced and evaluated. The thesis started by framing the problem of sparse rewards in the **RL** framework, proposing to address it by focusing on *exploration*. Doing so allows to efficiently analyze the whole search space and discover any obtainable reward, giving us the opportunity to later exploit it. At the same time, the search space needs to be properly defined in order to perform efficient exploration and discover these rewards. Collecting the necessary prior information about the task to solve and designing the search space itself requires a lot of engineering effort. This limits the range of applicability and the generalization ability of said policy learning algorithms. For this reason, reducing the amount of prior information needed at design time can prove beneficial. As a step in this direction, methods that can autonomously learn the search space in an online fashion have been introduced.

An overview of the literature on the subject of sparse rewards in Chapter 2. It starts by describing the framework of **RL** and how sparse reward problems can affect the performances of **RL** algorithms. The best case scenario in which to apply **RL** algorithms is in fact one in which the reward is dense. If it is not the case, other strategies need to be considered. A technique usually employed in these situations is to have the algorithm generate its own rewards. However, this can be complicated or lead the search to get stuck on local minima, as for example in the noisy TV problem discussed in Sec. 7.1.1. In light of this, the work focused on **EAs** as an alternative to **RL** for learning policies in situations of sparse rewards. Being episode-based rather than step-based means that **EAs** can more naturally deal with sparse rewards settings compared to **RL** methods. This is due to **EAs** needing the reward only at the end of the policy evaluation and not after each single action. At the same time, there can be situations in which the reward is extremely sparse and the policies do not get any reward. To deal with this, the *divergent search* family of algorithms has been considered. These **EAs** mainly focus on exploration. They do so by looking for a set of policies covering a given search space, called **Behaviour Space (BS)**, as much as possible. Among all the possible divergent search **EAs**, the work focused mainly on **NS**. The reason behind this choice lies in the ability of this method to quickly explore the whole **BS**, tending towards an uniform coverage of it. All of this is done while completely ignoring any reward and without the discretization of the **BS** required by other methods in



the same family.

Notwithstanding the great exploration abilities of **NS**, it is still limited by two main shortcomings. First, the search space needs to be hand-designed, which requires a lot of effort from the designer point of view, while also limiting the range of problems to which this algorithm can be applied. Second, by completely discarding rewards, **NS** ignores important information that can be useful for solving the task at hand. This thesis started by addressing the two issues separately. This has been done with the introduction of two algorithms: **TAXONS** and **SERENE**. Once proven the efficacy of these methods, an approach merging the complementary aspects of **TAXONS** and **SERENE** in a single algorithm was introduced: **STAX**. **STAX** can thus perform exploration of the **BS** while exploiting any discovered reward, with minimal prior information. This allows to address both limitations of **NS** at the same time. Let us recap how each of the mentioned methods work and what contributions they brought.

## **TAXONS**

**TAXONS**, introduced in Chapter 3, addresses the issue of hand-defining the search space in which **NS** evaluates the novelty of each policy. It does so by autonomously learning said space during the search process. This is done by taking advantage of the representation capabilities of an **AE**, while performing the search directly in its learned feature space. This space is learned online on the data generated by the policies during the search itself. The results show how, by learning the **BS** directly from high-dimensional RGB observation of the environment, it is possible to perform exploration almost as well as **NS**. Doing so greatly reduces the amount of prior information needed. At the same time, **TAXONS** relies on the assumption that the last image of the trajectory contains enough information to extract the behavior descriptor of a policy. In this regard, Chapter 4 discussed the Signature transform as a possible way to remove this assumption by representing the whole trajectory as a single vector. The aim was to be able to combine this method with **TAXONS** in order to calculate the behavior descriptor of a policy from the trajectory of high-dimensional RGB observations. This approach was compared against the simpler strategy of obtaining the behavior descriptor by just stacking few observations sampled along the trajectory. Eventually, this simpler strategy proved as effective as the more complex signature transform. The possible reason behind this is the high dimensionality of the descriptor extracted by the signature. Euclidean distances on high-dimensional vectors lose most of their relevance, lowering the effectiveness of the novelty metric in pushing for exploration.

## **SERENE**

The second limitation of **NS**, the discarding of any information coming from the reward, has been addressed with the introduction of **SERENE** in Chapter 5. This algorithm performs exploration through **NS**, then once rewards are

discovered it deploys emitters to exploit them. As explained in Sec. 5.2, emitters are instances of reward-based algorithms performing local search around the policies used to initialize the emitters themselves. This local search allows to quickly optimize on the reward without interfering with the exploration process. To this end, **SERENE** keeps the exploration of the search space and the exploitation of the rewards separated through a two alternating-steps process. This strategy allows the method to find a good compromise for the exploration-exploitation trade-off. **SERENE** has been shown to perform exploration in a fashion similar to **NS**. At the same time, it can efficiently optimize the reward for each discovered area, overcoming **NS**'s limitation. The reward optimization has been shown to be more efficient than other **QD** algorithms as **ME** or CMA-ME that perform exploration and reward exploitation at the same time. This proves that the separation of the exploration process from the exploitation can be advantageous in situations of particularly sparse rewards.

## STAX

As said, **TAXONS** and **SERENE** address two different problems of **NS**. These two approaches can be considered complementary in dealing with **NS** limitations and thus combined. This has been done in Chapter 6 with the introduction of the **STAX** algorithm. The method augments **SERENE** with **TAXONS** by using the latter to perform the search in the exploration step of the former. This strategy allows **STAX** to explore an unknown search space without relying on any prior information about the task. Then, once a reward is discovered, an emitter is initialized and evaluated to exploit said reward. At the same time, the **TAXONS**'s requirement for the last observation of the trajectory to be informative enough with respect to the whole policy behavior has been removed in **STAX**. This has been done by taking advantage of the lessons learned in Chapter 4. Multiple high-dimensional observations are sampled along a trajectory and their low-dimensional representations, generated by the **AE**, are stacked to form the behavior descriptor of the policy. The results show how this approach can foster exploration on levels similar to **NS**, with the minimal amount of prior information given to the algorithm. All of this without any loss in performance on the exploitation of the rewards. **STAX** is in fact able to discover and quickly exploit all reward areas present in the environment.

The advantages and contributions introduced by the methods developed during this thesis have been discussed in Chapter 7. Starting from **NS** and developing first **TAXONS** then **SERENE**, highlighted a path towards the introduction of a family of algorithms that can seamlessly deal with sparse reward environments. All the while requiring minimal knowledge about the task. This could potentially lead to the introduction of algorithms capable of great generalization, reducing to the minimum the intervention of the engineer when moving from one setting to the other. The introduction of **STAX** represents a promising step in this direction, unifying the two aspects of learning the **Behaviour Space (BS)** and exploiting any discovered reward. Nonetheless, the

path towards a solution of the sparse rewards problem is still long. Many improvements and extensions can be developed to augment the ability of [STAX](#), and [QD](#) algorithms in general, to better address this problem. At the same time, while framing the problem of sparse rewards from an [RL](#) point of view, we did not take advantage of the many innovations introduced in the field of [RL](#). We believe that a lot can be learned from the union of the two fields of [RL](#) and divergent search [EAs](#), thanks to the complementarity of the two approaches. This could lead to the development of many innovative approaches, the solution of many of the still unresolved problems in both fields and the application of learning algorithm to new and exciting fields of research.





---

# Bibliography

## References for Chapter 1: Introduction

- [1] Paul Guillaume. *Manuel de psychologie*. Presses universitaires de France, 1950 *Cited on page 1.*
- [2] Irenäus Eibl-Eibesfeldt. *Éthologie: biologie du comportement*. Editions OPHRYS, 1984 *Cited on page 1.*
- [3] Charles Darwin's. "On the origin of species". In: *published on 24* (1859) *Cited on pages 1, 4, 18.*
- [4] Michael Stephen Silk. *Homer: The Iliad*. Cambridge University Press, 2004 *Cited on page 1.*
- [5] Joseph Needham and Colin A Ronan. *The Shorter Science and Civilisation in China: Volume 2*. Vol. 2. Cambridge University Press, 1978 *Cited on page 1.*
- [6] Jessica Riskin. *Genesis redux: Essays in the history and philosophy of artificial life*. University of Chicago Press, 2010 *Cited on page 1.*
- [7] Dominik Zunt. "Who did actually invent the word "robot" and what does it mean?" In: *The Karel Čapek website* (2002) *Cited on page 2.*
- [8] Johanna Wallén. *The history of the industrial robot*. Linköping University Electronic Press, 2008 *Cited on page 2.*
- [9] Balkeshwar Singh, N Sellappan, and P Kumaradhas. "Evolution of industrial robots and their applications". In: *International Journal of emerging technology and advanced engineering* 3.5 (2013), pp. 763–768 *Cited on page 2.*
- [10] Martin Hägele et al. "Industrial robotics". In: *Springer handbook of robotics*. Springer, 2016, pp. 1385–1422 *Cited on page 2.*
- [11] Matt Simon. "The WIRED Guide to Robots". In: () *Cited on page 2.*
- [12] Andreas Kroll. "A survey on mobile robots for industrial inspection". In: *Proceedings of the Int. Conf. on Intelligent Autonomous Systems IAS10, Baden-Baden, Germany*. 2008, pp. 406–414 *Cited on page 2.*
- [13] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018 *Cited on pages 2, 10, 11, 13, 14, 112.*

- [14] Elliot A. Ludvig. “Reinforcement Learning in Animals”. In: *Encyclopedia of the Sciences of Learning*. Ed. by Norbert M. Seel. Boston, MA: Springer US, 2012, pp. 2799–2802. ISBN: 978-1-4419-1428-6. DOI: [10.1007/978-1-4419-1428-6\\_508](https://doi.org/10.1007/978-1-4419-1428-6_508) Cited on page 2.
- [15] Edward L Thorndike. “The law of effect”. In: *The American journal of psychology* 39.1/4 (1927), pp. 212–222 Cited on page 2.
- [16] IP Pavlov. “Conditioned Reflexes Dover Publications”. In: *Inc. (New York: Mineola)* (1927) Cited on page 2.
- [17] Encyclopaedia Britannica et al. *Britannica concise encyclopedia*. Encyclopaedia Britannica, Inc., 2008 Cited on page 2.
- [18] Hongzi Mao et al. “Resource management with deep reinforcement learning”. In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM. 2016, pp. 50–56 Cited on pages 3, 11.
- [19] Itamar Arel et al. “Reinforcement learning-based multi-agent system for network traffic signal control”. In: *IET Intelligent Transport Systems* 4.2 (2010), pp. 128–135 Cited on pages 3, 11.
- [20] Sergey Levine et al. “End-to-end training of deep visuomotor policies”. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373 Cited on pages 3, 11.
- [21] Zhenpeng Zhou, Xiaocheng Li, and Richard N Zare. “Optimizing chemical reactions with deep reinforcement learning”. In: *ACS central science* 3.12 (2017), pp. 1337–1344 Cited on pages 3, 11.
- [22] Robin R. Murphy et al. “Search and Rescue Robotics”. In: *Springer Handbook of Robotics* (2008), pp. 1151–1173 Cited on page 3.
- [23] Maja J Mataric. “Reward functions for accelerated learning”. In: *Machine learning proceedings 1994*. Elsevier, 1994, pp. 181–189 Cited on pages 3, 14.
- [24] Deepak Pathak et al. “Curiosity-driven exploration by self-supervised prediction”. In: *International conference on machine learning*. PMLR. 2017, pp. 2778–2787 Cited on pages 3, 17.
- [25] Yuri Burda et al. “Large-scale study of curiosity-driven learning”. In: *arXiv preprint arXiv:1808.04355* (2018) Cited on pages 3, 17.
- [26] Marcin Andrychowicz et al. “Hindsight experience replay”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5048–5058 Cited on pages 3, 15.
- [27] Max Jaderberg et al. “Reinforcement learning with unsupervised auxiliary tasks”. In: *arXiv preprint arXiv:1611.05397* (2016) Cited on page 3.

- [28] Pradnya A Vikhar. “Evolutionary algorithms: A critical review and its future prospects”. In: *2016 International conference on global trends in signal processing, information computing and communication (ICGT-SPICC)*. IEEE. 2016, pp. 261–265 *Cited on page 4.*
- [29] Kenneth De Jong. “Evolutionary computation: a unified approach”. In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*. 2016, pp. 185–199 *Cited on page 4.*
- [30] James Cohoon, John Kairo, and Jens Lienig. “Evolutionary algorithms for the physical design of VLSI circuits”. In: *Advances in Evolutionary Computing*. Springer, 2003, pp. 683–711 *Cited on page 4.*
- [31] E Gent. “Artificial intelligence is evolving all by itself”. In: *can be found under <https://www.sciencemag.org/news/2020/04/artificial-intelligence-evolving-all-itself>* (2020) *Cited on page 4.*
- [32] Edwin D De Jong and Jordan B Pollack. “Multi-objective methods for tree size control”. In: *Genetic Programming and Evolvable Machines* 4.3 (2003), pp. 211–233 *Cited on pages 4, 22.*
- [33] Khaled MS Badran and Peter I Rockett. “The roles of diversity preservation and mutation in preventing population collapse in multiobjective genetic programming”. In: *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. 2007, pp. 1551–1558 *Cited on pages 4, 22.*
- [34] Joel Lehman and Kenneth O Stanley. “Exploiting open-endedness to solve problems through the search for novelty.” In: *ALIFE*. 2008, pp. 329–336 *Cited on pages 4, 5, 23, 38, 39, 41, 72, 74, 88, 91.*
- [35] Joel Lehman and Kenneth O Stanley. “Abandoning objectives: Evolution through the search for novelty alone”. In: *Evolutionary computation* 19.2 (2011), pp. 189–223 *Cited on pages 5, 22.*
- [36] Jorge Gomes, Sancho Moura Oliveira, and Anders Lyhne Christensen. “An approach to evolve and exploit repertoires of general robot behaviours”. In: *Swarm and Evolutionary Computation* 43 (2018), pp. 265–283 *Cited on pages 5, 32.*
- [37] Konstantinos Chatzilygeroudis, Vassilis Vassiliades, and Jean-Baptiste Mouret. “Reset-free trial-and-error learning for robot damage recovery”. In: *Robotics and Autonomous Systems* 100 (2018), pp. 236–250 *Cited on pages 5, 32.*
- [38] Miguel Duarte et al. “Evolution of repertoire-based control for robots with complex locomotor systems”. In: *IEEE Transactions on Evolutionary Computation* 22.2 (2017), pp. 314–328 *Cited on pages 5, 32.*

- [39] Antoine Cully and J-B Mouret. “Evolving a behavioral repertoire for a walking robot”. In: *Evolutionary computation* 24.1 (2016), pp. 59–88  
*Cited on pages 5, 32.*
- [40] Joel Lehman and Kenneth O Stanley. “Evolving a diversity of virtual creatures through novelty search and local competition”. In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM. 2011, pp. 211–218  
*Cited on pages 5, 20, 25.*
- [41] Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. “Quality diversity: A new frontier for evolutionary computation”. In: *Frontiers in Robotics and AI* 3 (2016), p. 40  
*Cited on pages 5, 25, 64.*
- [42] Antoine Cully and Yiannis Demiris. “Quality and diversity optimization: A unifying modular framework”. In: *IEEE Transactions on Evolutionary Computation* 22.2 (2017), pp. 245–259  
*Cited on pages 5, 25.*
- [43] Jean-Baptiste Mouret and Jeff Clune. “Illuminating search spaces by mapping elites”. In: *arXiv preprint arXiv:1504.04909* (2015)  
*Cited on pages 5, 26, 57, 64, 74, 76, 91.*
- [44] Benjamin Eysenbach et al. “Diversity is all you need: Learning skills without a reward function”. In: *arXiv preprint arXiv:1802.06070* (2018)  
*Cited on pages 5, 25, 31.*
- [45] Daniele Gravina, Antonios Liapis, and Georgios Yannakakis. “Surprise search: Beyond objectives and novelty”. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. ACM. 2016, pp. 677–684  
*Cited on pages 5, 25.*
- [46] Antoine Cully et al. “Robots that can adapt like animals”. In: *Nature* 521.7553 (2015), p. 503  
*Cited on pages 5, 20, 25–27, 31, 33.*
- [47] Seungsu Kim and Stéphane Doncieux. “Learning highly diverse robot throwing movements through quality diversity search”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2017, pp. 1177–1178  
*Cited on page 6.*
- [48] Giuseppe Paolo et al. “Unsupervised Learning and Exploration of Reachable Outcome Space”. In: *algorithms* 24 (2019), p. 25  
*Cited on pages 6, 34, 57, 76, 91.*
- [49] Geoffrey E Hinton and Richard S Zemel. “Autoencoders, minimum description length and Helmholtz free energy”. In: *Advances in neural information processing systems*. 1994, pp. 3–10  
*Cited on pages 6, 32, 35.*
- [50] Kuo-Tsai Chen. “Iterated integrals and exponential homomorphisms”. In: *Proceedings of the London Mathematical Society* 3.1 (1954), pp. 502–512  
*Cited on pages 6, 49.*
- [51] Kuo-Tsai Chen. “Integration of paths, geometric invariants and a generalized Baker-Hausdorff formula”. In: *Annals of Mathematics* (1957), pp. 163–178  
*Cited on pages 6, 49.*



- [52] Kuo-Tsai Chen. “Integration of paths—A faithful representation of paths by noncommutative formal power series”. In: *Transactions of the American Mathematical Society* 89.2 (1958), pp. 395–407 Cited on pages 7, 49.
- [53] Patric Bonnier et al. “Deep signature transforms”. In: *Neural Information Processing Systems Foundation (NeurIPS)* (2019) Cited on pages 7, 49, 50, 53, 60.
- [54] Adeline Fermanian. “Embedding and learning with signatures”. In: *Computational Statistics & Data Analysis* 157 (2021), p. 107148 Cited on pages 7, 49, 53, 60, 109.
- [55] Matthew C Fontaine et al. “Covariance matrix adaptation for the rapid illumination of behavior space”. In: *Proceedings of the 2020 genetic and evolutionary computation conference*. 2020, pp. 94–102 Cited on pages 7, 28, 64, 65, 69, 74, 80, 112.
- [56] Antoine Cully. “Multi-Emitter MAP-Elites: Improving quality, diversity and convergence speed with heterogeneous sets of emitters”. In: *arXiv preprint arXiv:2007.05352* (2020) Cited on pages 7, 28, 64, 65, 69, 80, 106, 112.

## References for Chapter 2: Background and related work

- [3] Charles Darwin’s. “On the origin of species”. In: *published on* 24 (1859) Cited on pages 1, 4, 18.
- [13] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018 Cited on pages 2, 10, 11, 13, 14, 112.
- [18] Hongzi Mao et al. “Resource management with deep reinforcement learning”. In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM. 2016, pp. 50–56 Cited on pages 3, 11.
- [19] Itamar Arel et al. “Reinforcement learning-based multi-agent system for network traffic signal control”. In: *IET Intelligent Transport Systems* 4.2 (2010), pp. 128–135 Cited on pages 3, 11.
- [20] Sergey Levine et al. “End-to-end training of deep visuomotor policies”. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373 Cited on pages 3, 11.
- [21] Zhenpeng Zhou, Xiaocheng Li, and Richard N Zare. “Optimizing chemical reactions with deep reinforcement learning”. In: *ACS central science* 3.12 (2017), pp. 1337–1344 Cited on pages 3, 11.
- [23] Maja J Mataric. “Reward functions for accelerated learning”. In: *Machine learning proceedings 1994*. Elsevier, 1994, pp. 181–189 Cited on pages 3, 14.

- [24] Deepak Pathak et al. “Curiosity-driven exploration by self-supervised prediction”. In: *International conference on machine learning*. PMLR. 2017, pp. 2778–2787 *Cited on pages 3, 17.*
- [25] Yuri Burda et al. “Large-scale study of curiosity-driven learning”. In: *arXiv preprint arXiv:1808.04355* (2018) *Cited on pages 3, 17.*
- [26] Marcin Andrychowicz et al. “Hindsight experience replay”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5048–5058 *Cited on pages 3, 15.*
- [32] Edwin D De Jong and Jordan B Pollack. “Multi-objective methods for tree size control”. In: *Genetic Programming and Evolvable Machines* 4.3 (2003), pp. 211–233 *Cited on pages 4, 22.*
- [33] Khaled MS Badran and Peter I Rockett. “The roles of diversity preservation and mutation in preventing population collapse in multiobjective genetic programming”. In: *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. 2007, pp. 1551–1558 *Cited on pages 4, 22.*
- [34] Joel Lehman and Kenneth O Stanley. “Exploiting open-endedness to solve problems through the search for novelty.” In: *ALIFE*. 2008, pp. 329–336 *Cited on pages 4, 5, 23, 38, 39, 41, 72, 74, 88, 91.*
- [35] Joel Lehman and Kenneth O Stanley. “Abandoning objectives: Evolution through the search for novelty alone”. In: *Evolutionary computation* 19.2 (2011), pp. 189–223 *Cited on pages 5, 22.*
- [40] Joel Lehman and Kenneth O Stanley. “Evolving a diversity of virtual creatures through novelty search and local competition”. In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM. 2011, pp. 211–218 *Cited on pages 5, 20, 25.*
- [41] Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. “Quality diversity: A new frontier for evolutionary computation”. In: *Frontiers in Robotics and AI* 3 (2016), p. 40 *Cited on pages 5, 25, 64.*
- [42] Antoine Cully and Yiannis Demiris. “Quality and diversity optimization: A unifying modular framework”. In: *IEEE Transactions on Evolutionary Computation* 22.2 (2017), pp. 245–259 *Cited on pages 5, 25.*
- [43] Jean-Baptiste Mouret and Jeff Clune. “Illuminating search spaces by mapping elites”. In: *arXiv preprint arXiv:1504.04909* (2015) *Cited on pages 5, 26, 57, 64, 74, 76, 91.*
- [44] Benjamin Eysenbach et al. “Diversity is all you need: Learning skills without a reward function”. In: *arXiv preprint arXiv:1802.06070* (2018) *Cited on pages 5, 25, 31.*

- [45] Daniele Gravina, Antonios Liapis, and Georgios Yannakakis. “Surprise search: Beyond objectives and novelty”. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. ACM, 2016, pp. 677–684 *Cited on pages 5, 25.*
- [46] Antoine Cully et al. “Robots that can adapt like animals”. In: *Nature* 521.7553 (2015), p. 503 *Cited on pages 5, 20, 25–27, 31, 33.*
- [55] Matthew C Fontaine et al. “Covariance matrix adaptation for the rapid illumination of behavior space”. In: *Proceedings of the 2020 genetic and evolutionary computation conference*. 2020, pp. 94–102 *Cited on pages 7, 28, 64, 65, 69, 74, 80, 112.*
- [56] Antoine Cully. “Multi-Emitter MAP-Elites: Improving quality, diversity and convergence speed with heterogeneous sets of emitters”. In: *arXiv preprint arXiv:2007.05352* (2020) *Cited on pages 7, 28, 64, 65, 69, 80, 106, 112.*
- [57] Jens Kober and Jan Peters. “Policy search for motor primitives in robotics”. In: *Learning Motor Skills*. Springer, 2014, pp. 83–117 *Cited on page 10.*
- [58] Jens Kober, J Andrew Bagnell, and Jan Peters. “Reinforcement learning in robotics: A survey”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274 *Cited on page 10.*
- [59] Olivier Sigaud and Freek Stulp. “Policy search in continuous action domains: an overview”. In: *Neural Networks* 113 (2019), pp. 28–40 *Cited on pages 10, 13.*
- [60] Richard Bellman. “A Markovian Decision Process”. In: *Indiana Univ. Math. J.* 6 (4 1957), pp. 679–684. ISSN: 0022-2518 *Cited on page 11.*
- [61] Ronald A Howard. “Dynamic programming and markov processes.” In: (1960) *Cited on page 11.*
- [62] Michael N Katehakis and Arthur F Veinott Jr. “The multi-armed bandit problem: decomposition and computation”. In: *Mathematics of Operations Research* 12.2 (1987), pp. 262–268 *Cited on page 11.*
- [63] Aleksandrs Slivkins. “Introduction to multi-armed bandits”. In: *arXiv preprint arXiv:1904.07272* (2019) *Cited on pages 13, 17, 112.*
- [64] Richard Bellman. *An introduction to the theory of dynamic programming*. Tech. rep. RAND CORP SANTA MONICA CA, 1953 *Cited on page 13.*
- [65] Adam Sykulski. “The exploration-exploitation trade-off in sequential decision making problems”. PhD thesis. Imperial College Press, 2011 *Cited on page 13.*
- [66] Joshua Hare. “Dealing with sparse rewards in reinforcement learning”. In: *arXiv preprint arXiv:1910.09281* (2019) *Cited on page 14.*

- [67] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: [arXiv:1606.01540](https://arxiv.org/abs/1606.01540)  
Cited on page 15.
- [68] Yujing Hu et al. “Learning to utilize shaping rewards: A new approach of reward shaping”. In: *arXiv preprint arXiv:2011.02669* (2020) Cited on page 15.
- [69] Babak Badnava and Nasser Mozayani. “A new potential-based reward shaping for reinforcement learning agent”. In: *arXiv preprint arXiv:1902.06239* (2019) Cited on page 15.
- [70] Christopher Berner et al. “Dota 2 with large scale deep reinforcement learning”. In: *arXiv preprint arXiv:1912.06680* (2019) Cited on page 15.
- [71] Alexander Trott et al. “Keeping your distance: Solving sparse reward tasks using self-balancing shaped rewards”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 10376–10386 Cited on page 15.
- [72] Rico Jonschkowski and Oliver Brock. “State Representation Learning in Robotics: Using Prior Knowledge about Physical Interaction.” In: *Robotics: Science and Systems*. 2014 Cited on page 15.
- [73] *Dota 2*. <https://www.dota2.com/home> Cited on page 15.
- [74] Andrew Levy et al. “Learning multi-level hierarchies with hindsight”. In: *arXiv preprint arXiv:1712.00948* (2017) Cited on page 15.
- [75] Ashvin V Nair et al. “Visual reinforcement learning with imagined goals”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 9191–9200 Cited on page 15.
- [76] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013) Cited on page 15.
- [77] Carlos Florensa et al. “Automatic goal generation for reinforcement learning agents”. In: *International conference on machine learning*. PMLR. 2018, pp. 1515–1528 Cited on page 16.
- [78] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems 27* (2014) Cited on page 16.
- [79] Martin Riedmiller et al. “Learning by playing solving sparse reward tasks from scratch”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4344–4353 Cited on page 16.
- [80] Pierre-Yves Oudeyer and Frederic Kaplan. “What is intrinsic motivation? A typology of computational approaches”. In: *Frontiers in neurobotics 1* (2009), p. 6 Cited on page 16.
- [81] Arthur Aubret, Laetitia Matignon, and Salima Hassas. “A survey on intrinsic motivation in reinforcement learning”. In: *arXiv preprint arXiv:1908.06976* (2019) Cited on pages 16, 17.

- [82] Alison Gopnik, Andrew N Meltzoff, and Patricia K Kuhl. *The scientist in the crib: Minds, brains, and how children learn*. William Morrow & Co, 1999 *Cited on page 16.*
- [83] Satinder Singh, Richard L Lewis, and Andrew G Barto. “Where do rewards come from”. In: *Proceedings of the annual conference of the cognitive science society*. Cognitive Science Society. 2009, pp. 2601–2606 *Cited on page 16.*
- [84] Ronen I Brafman and Moshe Tennenholtz. “R-max-a general polynomial time algorithm for near-optimal reinforcement learning”. In: *Journal of Machine Learning Research* 3.Oct (2002), pp. 213–231 *Cited on page 16.*
- [85] Michael Kearns and Satinder Singh. “Near-optimal reinforcement learning in polynomial time”. In: *Machine learning* 49.2 (2002), pp. 209–232 *Cited on page 16.*
- [86] Marc Bellemare et al. “Unifying count-based exploration and intrinsic motivation”. In: *Advances in Neural Information Processing Systems*. Vol. 29. 2016, pp. 1471–1479 *Cited on page 16.*
- [87] Yuri Burda et al. “Exploration by random network distillation”. In: *arXiv preprint arXiv:1810.12894* (2018) *Cited on pages 16, 37, 110.*
- [88] Alexander S Klyubin, Daniel Polani, and Chrystopher L Nehaniv. “Empowerment: A universal agent-centric measure of control”. In: *2005 ieee congress on evolutionary computation*. Vol. 1. IEEE. 2005, pp. 128–135 *Cited on page 17.*
- [89] Alexander S Klyubin, Daniel Polani, and Chrystopher L Nehaniv. “All else being equal be empowered”. In: *European Conference on Artificial Life*. Springer. 2005, pp. 744–753 *Cited on page 17.*
- [90] Christoph Salge, Cornelius Glackin, and Daniel Polani. “Empowerment—an introduction”. In: *Guided Self-Organization: Inception*. Springer, 2014, pp. 67–114 *Cited on page 17.*
- [91] James V Stone. “Information theory: a tutorial introduction”. In: (2015) *Cited on page 17.*
- [92] Shakir Mohamed and Danilo Jimenez Rezende. “Variational information maximisation for intrinsically motivated reinforcement learning”. In: *arXiv preprint arXiv:1509.08731* (2015) *Cited on page 17.*
- [93] Pierre-Yves Oudeyer. “Intelligent adaptive curiosity: a source of self-development”. In: (2004) *Cited on page 17.*
- [94] Paul J Silvia. “Curiosity and motivation”. In: *The Oxford handbook of human motivation* (2012), pp. 157–166 *Cited on page 17.*
- [95] Pierre-Yves Oudeyer. “Computational theories of curiosity-driven learning”. In: *arXiv preprint arXiv:1802.10546* (2018) *Cited on page 17.*

- [96] Adrien Baranes and Pierre-Yves Oudeyer. “Active learning of inverse models with intrinsically motivated goal exploration in robots”. In: *Robotics and Autonomous Systems* 61.1 (2013), pp. 49–73 *Cited on page 17.*
- [97] Sébastien Forestier, Yoan Mollard, and Pierre-Yves Oudeyer. “Intrinsically motivated goal exploration processes with automatic curriculum learning”. In: *arXiv preprint arXiv:1708.02190* (2017) *Cited on pages 17, 18.*
- [98] Adrien Laversanne-Finot, Alexandre Pere, and Pierre-Yves Oudeyer. “Curiosity driven exploration of learned disentangled goal spaces”. In: *Conference on Robot Learning*. PMLR. 2018, pp. 487–504 *Cited on pages 17, 110.*
- [99] Alexandre Péré et al. “Unsupervised learning of goal spaces for intrinsically motivated goal exploration”. In: *arXiv preprint arXiv:1803.00781* (2018) *Cited on page 18.*
- [100] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. “Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1039–1048 *Cited on pages 18, 65.*
- [101] Adrien Ecoffet et al. “Go-explore: a new approach for hard-exploration problems”. In: *arXiv preprint arXiv:1901.10995* (2019) *Cited on page 18.*
- [102] Melanie Mitchell. *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262631857 *Cited on page 18.*
- [103] Agoston E Eiben and Jim Smith. “From evolutionary computation to the evolution of things”. In: *Nature* 521.7553 (2015), pp. 476–482 *Cited on pages 18–20.*
- [104] Brad L Miller, David E Goldberg, et al. “Genetic algorithms, tournament selection, and the effects of noise”. In: *Complex systems* 9.3 (1995), pp. 193–212 *Cited on page 19.*
- [105] Antoine Cully. “Creative adaptation through learning”. PhD thesis. Paris 6, 2015 *Cited on page 20.*
- [106] Peter Schwehr. “Evolutionary algorithms in architecture”. In: *open house international* (2011) *Cited on page 20.*
- [107] Martin Dostál. “Evolutionary music composition”. In: *Handbook of Optimization*. Springer, 2013, pp. 935–964 *Cited on page 20.*
- [108] Jimmy Secretan et al. “Picbreeder: evolving pictures collaboratively online”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2008, pp. 1759–1768 *Cited on page 20.*
- [109] Eric Medvet et al. “Biodiversity in evolved voxel-based soft robots”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2021, pp. 129–137 *Cited on page 20.*

- [110] Kenneth O Stanley and Risto Miikkulainen. “Evolving neural networks through augmenting topologies”. In: *Evolutionary computation* 10.2 (2002), pp. 99–127 *Cited on pages 20, 22.*
- [111] Kenneth O Stanley et al. “Designing neural networks through neuroevolution”. In: *Nature Machine Intelligence* 1.1 (2019), pp. 24–35 *Cited on page 20.*
- [112] Viktor Zykov, Josh Bongard, and Hod Lipson. “Evolving dynamic gaits on a physical robot”. In: *Proceedings of Genetic and Evolutionary Computation Conference, Late Breaking Paper, GECCO*. Vol. 4. 2004 *Cited on page 20.*
- [113] Stephane Doncieux and Jean-Baptiste Mouret. “Behavioral diversity measures for evolutionary robotics”. In: *IEEE congress on evolutionary computation*. IEEE. 2010, pp. 1–8 *Cited on pages 20, 37, 56.*
- [114] C-L Hwang and Abu Syed Md Masud. *Multiple objective decision making—methods and applications: a state-of-the-art survey*. Vol. 164. Springer Science & Business Media, 2012 *Cited on pages 20, 21.*
- [115] Kalyanmoy Deb et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197 *Cited on pages 21, 23, 63, 74, 84, 91.*
- [116] Carlo R Raquel and Prospero C Naval Jr. “An effective use of crowding distance in multiobjective particle swarm optimization”. In: *Proceedings of the 7th Annual conference on Genetic and Evolutionary Computation*. 2005, pp. 257–264 *Cited on pages 21, 22.*
- [117] Mikkel T Jensen. “Helper-objectives: Using multi-objective evolutionary algorithms for single-objective optimisation”. In: *Journal of Mathematical Modelling and Algorithms* 3.4 (2004), pp. 323–347 *Cited on page 22.*
- [118] Stephane Doncieux and Jean-Baptiste Mouret. “Beyond black-box optimization: a review of selective pressures for evolutionary robotics”. In: *Evolutionary Intelligence* 7.2 (2014), pp. 71–93 *Cited on page 22.*
- [119] Samir W Mahfoud. “Nicheing methods for genetic algorithms”. PhD thesis. University of Illinois at Urbana-Champaign, 1995 *Cited on page 22.*
- [120] Xiaodong Yin and Noel Garmay. “A fast genetic algorithm with sharing scheme using cluster analysis methods in multimodal function optimization”. In: *Artificial neural nets and genetic algorithms*. Springer. 1993, pp. 450–457 *Cited on page 22.*
- [121] Georges R Harik et al. “Finding Multimodal Solutions Using Restricted Tournament Selection.” In: *ICGA*. Citeseer. 1995, pp. 24–31 *Cited on page 23.*



- [122] Joel Lehman, Sebastian Risi, Kenneth O Stanley, et al. “On the benefits of divergent search for evolved representations”. In: *Proceedings of the EvoNet 2012 Workshop at ALIFE XIII*. 2012 Cited on page 23.
- [123] J-B Mouret and Stéphane Doncieux. “Encouraging behavioral diversity in evolutionary robotics: An empirical study”. In: *Evolutionary computation* 20.1 (2012), pp. 91–133 Cited on page 23.
- [124] Jorge Gomes, Pedro Mariano, and Anders Lyhne Christensen. “Devising effective novelty search algorithms: A comprehensive empirical study”. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 943–950 Cited on pages 24, 40.
- [125] Stephane Doncieux, Alban Laflaquière, and Alexandre Coninx. “Novelty search: a theoretical perspective”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2019, pp. 99–106 Cited on pages 25, 32, 63.
- [126] Christopher Stanton and Jeff Clune. “Curiosity search: producing generalists by encouraging individuals to continually explore and acquire skills throughout their lifetime”. In: *PloS one* 11.9 (2016), e0162235 Cited on page 25.
- [127] Víctor Campos et al. “Explore, Discover and Learn: Unsupervised Discovery of State-Covering Skills”. In: *arXiv preprint arXiv:2002.03647* (2020) Cited on page 25.
- [128] Jerome H Friedman, Jon Louis Bentley, and Raphael A Finkel. *An algorithm for finding best matches in logarithmic time*. Department of Computer Science, Stanford University, 1975 Cited on page 26.
- [129] Vassilis Vassiliades, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. “Scaling up map-elites using centroidal voronoi tessellations”. In: *arXiv preprint arXiv:1610.05729* (2016) Cited on pages 27, 28.
- [130] Niels Justesen, Sebastian Risi, and Jean-Baptiste Mouret. “Map-elites for noisy domains by adaptive sampling”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2019, pp. 121–122 Cited on pages 28, 113.
- [131] Manon Flageat and Antoine Cully. “Fast and stable MAP-Elites in noisy domains using deep grids”. In: *Artificial Life Conference Proceedings*. MIT Press. 2020, pp. 273–282 Cited on pages 28, 113.
- [132] Hans-Georg Beyer and Hans-Paul Schwefel. “Evolution strategies—A comprehensive introduction”. In: *Natural computing* 1.1 (2002), pp. 3–52 Cited on page 28.
- [133] Tim Salimans et al. “Evolution strategies as a scalable alternative to reinforcement learning”. In: *arXiv preprint arXiv:1703.03864* (2017) Cited on page 28.



- [134] Alois Pourchot and Olivier Sigaud. “CEM-RL: Combining evolutionary and gradient-based methods for policy search”. In: *arXiv preprint arXiv:1810.01222* (2018) *Cited on page 28.*
- [135] Geoffrey Cideron et al. “QD-RL: Efficient Mixing of Quality and Diversity in Reinforcement Learning”. In: *arXiv preprint arXiv:2006.08505* (2020) *Cited on pages 28, 73.*

## References for Chapter 3: TAXONS

- [34] Joel Lehman and Kenneth O Stanley. “Exploiting open-endedness to solve problems through the search for novelty.” In: *ALIFE*. 2008, pp. 329–336 *Cited on pages 4, 5, 23, 38, 39, 41, 72, 74, 88, 91.*
- [36] Jorge Gomes, Sancho Moura Oliveira, and Anders Lyhne Christensen. “An approach to evolve and exploit repertoires of general robot behaviours”. In: *Swarm and Evolutionary Computation* 43 (2018), pp. 265–283 *Cited on pages 5, 32.*
- [37] Konstantinos Chatzilygeroudis, Vassilis Vassiliades, and Jean-Baptiste Mouret. “Reset-free trial-and-error learning for robot damage recovery”. In: *Robotics and Autonomous Systems* 100 (2018), pp. 236–250 *Cited on pages 5, 32.*
- [38] Miguel Duarte et al. “Evolution of repertoire-based control for robots with complex locomotor systems”. In: *IEEE Transactions on Evolutionary Computation* 22.2 (2017), pp. 314–328 *Cited on pages 5, 32.*
- [39] Antoine Cully and J-B Mouret. “Evolving a behavioral repertoire for a walking robot”. In: *Evolutionary computation* 24.1 (2016), pp. 59–88 *Cited on pages 5, 32.*
- [44] Benjamin Eysenbach et al. “Diversity is all you need: Learning skills without a reward function”. In: *arXiv preprint arXiv:1802.06070* (2018) *Cited on pages 5, 25, 31.*
- [46] Antoine Cully et al. “Robots that can adapt like animals”. In: *Nature* 521.7553 (2015), p. 503 *Cited on pages 5, 20, 25–27, 31, 33.*
- [48] Giuseppe Paolo et al. “Unsupervised Learning and Exploration of Reachable Outcome Space”. In: *algorithms* 24 (2019), p. 25 *Cited on pages 6, 34, 57, 76, 91.*
- [49] Geoffrey E Hinton and Richard S Zemel. “Autoencoders, minimum description length and Helmholtz free energy”. In: *Advances in neural information processing systems*. 1994, pp. 3–10 *Cited on pages 6, 32, 35.*
- [87] Yuri Burda et al. “Exploration by random network distillation”. In: *arXiv preprint arXiv:1810.12894* (2018) *Cited on pages 16, 37, 110.*
- [113] Stephane Doncieux and Jean-Baptiste Mouret. “Behavioral diversity measures for evolutionary robotics”. In: *IEEE congress on evolutionary computation*. IEEE. 2010, pp. 1–8 *Cited on pages 20, 37, 56.*

- [124] Jorge Gomes, Pedro Mariano, and Anders Lyhne Christensen. “Devising effective novelty search algorithms: A comprehensive empirical study”. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 943–950 Cited on pages [24](#), [40](#).
- [125] Stephane Doncieux, Alban Laflaquière, and Alexandre Coninx. “Novelty search: a theoretical perspective”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2019, pp. 99–106 Cited on pages [25](#), [32](#), [63](#).
- [136] Giuseppe Paolo et al. “Unsupervised learning and exploration of reachable outcome space”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 2379–2385 Cited on page [32](#).
- [137] Timothée Lesort et al. “State representation learning for control: An overview”. In: *Neural Networks* 108 (2018), pp. 379–392 Cited on pages [32](#), [111](#).
- [138] Elliot Meyerson, Joel Lehman, and Risto Miikkulainen. “Learning behavior characterizations for novelty search”. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. ACM. 2016, pp. 149–156 Cited on page [33](#).
- [139] Antonios Liapis et al. “Transforming Exploratory Creativity with DeLeNoX,.” In: *ICCC*. 2013, pp. 56–63 Cited on page [33](#).
- [140] Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. “Innovation engines: Automated creativity and improved stochastic optimization via deep learning”. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 959–966 Cited on page [33](#).
- [141] Antoine Cully and Yiannis Demiris. “Hierarchical behavioral repertoires with unsupervised descriptors”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2018, pp. 69–76 Cited on pages [33](#), [113](#).
- [142] Antoine Cully. “Autonomous skill discovery with Quality-Diversity and Unsupervised Descriptors”. In: *GECCO 2019* (2019) Cited on pages [33](#), [86](#).
- [143] Charu Aggarwal, Alexander Hinneburg, and Daniel Keim. “On the Surprising Behavior of Distance Metric in High-Dimensional Space”. In: *First publ. in: Database theory, ICDT 200, 8th International Conference, London, UK, January 4 - 6, 2001 / Jan Van den Bussche ... (eds.). Berlin: Springer, 2001, pp. 420-434 (=Lecture notes in computer science ; 1973)* (Feb. 2002) Cited on pages [35](#), [49](#), [53](#), [109](#).

- [144] Carlos Oscar Sánchez Sorzano, Javier Vargas, and A Pascual Montano. “A survey of dimensionality reduction techniques”. In: *arXiv preprint arXiv:1403.2877* (2014) *Cited on page 35.*
- [145] Emanuel Todorov, Tom Erez, and Yuval Tassa. “Mujoco: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033 *Cited on page 38.*
- [146] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. <http://pybullet.org>. 2019 *Cited on pages 38, 42.*
- [147] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167* (2015) *Cited on page 40.*
- [148] Günter Klambauer et al. “Self-normalizing neural networks”. In: *Advances in neural information processing systems*. 2017, pp. 971–980 *Cited on pages 40, 92.*
- [149] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014) *Cited on pages 40, 92.*
- [150] Henry B Mann and Donald R Whitney. “On a test of whether one of two random variables is stochastically larger than the other”. In: *The annals of mathematical statistics* (1947), pp. 50–60 *Cited on page 41.*
- [151] Sture Holm. “A simple sequentially rejective multiple test procedure”. In: *Scandinavian journal of statistics* (1979), pp. 65–70 *Cited on page 41.*
- [152] Stefan Schaal. “Dynamic movement primitives—a framework for motor control in humans and humanoid robotics”. In: *Adaptive motion of animals and machines*. Springer, 2006, pp. 261–280 *Cited on page 41.*
- [153] Austin Stone et al. “The Distracting Control Suite—A Challenging Benchmark for Reinforcement Learning from Pixels”. In: *arXiv preprint arXiv:2101.02722* (2021) *Cited on pages 41, 110.*
- [154] John Schulman et al. “High-dimensional continuous control using generalized advantage estimation”. In: *arXiv preprint arXiv:1506.02438* (2015) *Cited on page 41.*
- [155] Andrew M Saxe et al. “On random weights and unsupervised feature learning.” In: *ICML*. Vol. 2. 3. 2011, p. 6 *Cited on page 45.*

## References for Chapter 4: Signatures

- [43] Jean-Baptiste Mouret and Jeff Clune. “Illuminating search spaces by mapping elites”. In: *arXiv preprint arXiv:1504.04909* (2015) *Cited on pages 5, 26, 57, 64, 74, 76, 91.*

- [48] Giuseppe Paolo et al. “Unsupervised Learning and Exploration of Reachable Outcome Space”. In: *algorithms* 24 (2019), p. 25 Cited on pages 6, 34, 57, 76, 91.
- [50] Kuo-Tsai Chen. “Iterated integrals and exponential homomorphisms”. In: *Proceedings of the London Mathematical Society* 3.1 (1954), pp. 502–512 Cited on pages 6, 49.
- [51] Kuo-Tsai Chen. “Integration of paths, geometric invariants and a generalized Baker-Hausdorff formula”. In: *Annals of Mathematics* (1957), pp. 163–178 Cited on pages 6, 49.
- [52] Kuo-Tsai Chen. “Integration of paths—A faithful representation of paths by noncommutative formal power series”. In: *Transactions of the American Mathematical Society* 89.2 (1958), pp. 395–407 Cited on pages 7, 49.
- [53] Patric Bonnier et al. “Deep signature transforms”. In: *Neural Information Processing Systems Foundation (NeurIPS)* (2019) Cited on pages 7, 49, 50, 53, 60.
- [54] Adeline Fermanian. “Embedding and learning with signatures”. In: *Computational Statistics & Data Analysis* 157 (2021), p. 107148 Cited on pages 7, 49, 53, 60, 109.
- [113] Stephane Doncieux and Jean-Baptiste Mouret. “Behavioral diversity measures for evolutionary robotics”. In: *IEEE congress on evolutionary computation*. IEEE. 2010, pp. 1–8 Cited on pages 20, 37, 56.
- [143] Charu Aggarwal, Alexander Hinneburg, and Daniel Keim. “On the Surprising Behavior of Distance Metric in High-Dimensional Space”. In: *First publ. in: Database theory, ICDT 200, 8th International Conference, London, UK, January 4 - 6, 2001 / Jan Van den Bussche ... (eds.). Berlin: Springer, 2001, pp. 420-434 (=Lecture notes in computer science ; 1973)* (Feb. 2002) Cited on pages 35, 49, 53, 109.
- [156] Meinard Müller. “Dynamic time warping”. In: *Information retrieval for music and motion* (2007), pp. 69–84 Cited on page 49.
- [157] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780 Cited on page 49.
- [158] Martin Hairer. “Solving the KPZ equation”. In: *Annals of Mathematics* (2013), pp. 559–664 Cited on page 50.
- [159] Weixin Yang et al. “Developing the path signature methodology and its application to landmark-based human action recognition”. In: *arXiv preprint arXiv:1707.03993* (2017) Cited on pages 50, 51, 53.
- [160] Benjamin Graham. “Sparse arrays of signatures for online character recognition”. In: *arXiv preprint arXiv:1308.0371* (2013) Cited on page 53.

## References for Chapter 5: SERENE

- [34] Joel Lehman and Kenneth O Stanley. “Exploiting open-endedness to solve problems through the search for novelty.” In: *ALIFE*. 2008, pp. 329–336  
*Cited on pages 4, 5, 23, 38, 39, 41, 72, 74, 88, 91.*
- [41] Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. “Quality diversity: A new frontier for evolutionary computation”. In: *Frontiers in Robotics and AI* 3 (2016), p. 40  
*Cited on pages 5, 25, 64.*
- [43] Jean-Baptiste Mouret and Jeff Clune. “Illuminating search spaces by mapping elites”. In: *arXiv preprint arXiv:1504.04909* (2015)  
*Cited on pages 5, 26, 57, 64, 74, 76, 91.*
- [48] Giuseppe Paolo et al. “Unsupervised Learning and Exploration of Reachable Outcome Space”. In: *algorithms* 24 (2019), p. 25  
*Cited on pages 6, 34, 57, 76, 91.*
- [55] Matthew C Fontaine et al. “Covariance matrix adaptation for the rapid illumination of behavior space”. In: *Proceedings of the 2020 genetic and evolutionary computation conference*. 2020, pp. 94–102  
*Cited on pages 7, 28, 64, 65, 69, 74, 80, 112.*
- [56] Antoine Cully. “Multi-Emitter MAP-Elites: Improving quality, diversity and convergence speed with heterogeneous sets of emitters”. In: *arXiv preprint arXiv:2007.05352* (2020)  
*Cited on pages 7, 28, 64, 65, 69, 80, 106, 112.*
- [100] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. “Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1039–1048  
*Cited on pages 18, 65.*
- [115] Kalyanmoy Deb et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197  
*Cited on pages 21, 23, 63, 74, 84, 91.*
- [125] Stephane Doncieux, Alban Laflaquière, and Alexandre Coninx. “Novelty search: a theoretical perspective”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2019, pp. 99–106  
*Cited on pages 25, 32, 63.*
- [135] Geoffrey Cideron et al. “QD-RL: Efficient Mixing of Quality and Diversity in Reinforcement Learning”. In: *arXiv preprint arXiv:2006.08505* (2020)  
*Cited on pages 28, 73.*
- [161] Giuseppe Paolo et al. “Sparse Reward Exploration via Novelty Search and Emitters”. In: *arXiv preprint arXiv:2102.03140* (2021)  
*Cited on pages 64, 88, 91.*
- [162] Nikolaus Hansen. “The CMA evolution strategy: A tutorial”. In: *arXiv preprint arXiv:1604.00772* (2016)  
*Cited on pages 64, 69, 70.*
- [163] Giuseppe Paolo. *Billiard*. <https://github.com/GPaolo/Billiard>. 2020  
*Cited on page 72.*

- [164] Pontus Loviken and Nikolas Hemion. “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)*. IEEE. 2017, pp. 247–254 Cited on pages 73, 90.
- [165] Jacqueline Gottlieb et al. “Information-seeking, curiosity, and attention: computational and neural mechanisms”. In: *Trends in cognitive sciences* 17.11 (2013), pp. 585–593 Cited on page 75.
- [166] Sebastian Blaes et al. “Control What You Can: Intrinsically Motivated Task-Planning Agent”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019 Cited on page 75.

## References for Chapter 6: STAX

- [34] Joel Lehman and Kenneth O Stanley. “Exploiting open-endedness to solve problems through the search for novelty.” In: *ALIFE*. 2008, pp. 329–336 Cited on pages 4, 5, 23, 38, 39, 41, 72, 74, 88, 91.
- [43] Jean-Baptiste Mouret and Jeff Clune. “Illuminating search spaces by mapping elites”. In: *arXiv preprint arXiv:1504.04909* (2015) Cited on pages 5, 26, 57, 64, 74, 76, 91.
- [48] Giuseppe Paolo et al. “Unsupervised Learning and Exploration of Reachable Outcome Space”. In: *algorithms* 24 (2019), p. 25 Cited on pages 6, 34, 57, 76, 91.
- [56] Antoine Cully. “Multi-Emitter MAP-Elites: Improving quality, diversity and convergence speed with heterogeneous sets of emitters”. In: *arXiv preprint arXiv:2007.05352* (2020) Cited on pages 7, 28, 64, 65, 69, 80, 106, 112.
- [115] Kalyanmoy Deb et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197 Cited on pages 21, 23, 63, 74, 84, 91.
- [142] Antoine Cully. “Autonomous skill discovery with Quality-Diversity and Unsupervised Descriptors”. In: *GECCO 2019* (2019) Cited on pages 33, 86.
- [148] Günter Klambauer et al. “Self-normalizing neural networks”. In: *Advances in neural information processing systems*. 2017, pp. 971–980 Cited on pages 40, 92.
- [149] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014) Cited on pages 40, 92.
- [161] Giuseppe Paolo et al. “Sparse Reward Exploration via Novelty Search and Emitters”. In: *arXiv preprint arXiv:2102.03140* (2021) Cited on pages 64, 88, 91.

- [164] Pontus Loviken and Nikolas Hemion. “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)*. IEEE. 2017, pp. 247–254 Cited on pages [73](#), [90](#).
- [167] Karl Pearson. “VII. Note on regression and inheritance in the case of two parents”. In: *proceedings of the royal society of London* 58.347-352 (1895), pp. 240–242 Cited on page [104](#).

## References for Chapter 7: Discussion

- [13] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018 Cited on pages [2](#), [10](#), [11](#), [13](#), [14](#), [112](#).
- [54] Adeline Fermanian. “Embedding and learning with signatures”. In: *Computational Statistics & Data Analysis* 157 (2021), p. 107148 Cited on pages [7](#), [49](#), [53](#), [60](#), [109](#).
- [55] Matthew C Fontaine et al. “Covariance matrix adaptation for the rapid illumination of behavior space”. In: *Proceedings of the 2020 genetic and evolutionary computation conference*. 2020, pp. 94–102 Cited on pages [7](#), [28](#), [64](#), [65](#), [69](#), [74](#), [80](#), [112](#).
- [56] Antoine Cully. “Multi-Emitter MAP-Elites: Improving quality, diversity and convergence speed with heterogeneous sets of emitters”. In: *arXiv preprint arXiv:2007.05352* (2020) Cited on pages [7](#), [28](#), [64](#), [65](#), [69](#), [80](#), [106](#), [112](#).
- [63] Aleksandrs Slivkins. “Introduction to multi-armed bandits”. In: *arXiv preprint arXiv:1904.07272* (2019) Cited on pages [13](#), [17](#), [112](#).
- [87] Yuri Burda et al. “Exploration by random network distillation”. In: *arXiv preprint arXiv:1810.12894* (2018) Cited on pages [16](#), [37](#), [110](#).
- [98] Adrien Laversanne-Finot, Alexandre Pere, and Pierre-Yves Oudeyer. “Curiosity driven exploration of learned disentangled goal spaces”. In: *Conference on Robot Learning*. PMLR. 2018, pp. 487–504 Cited on pages [17](#), [110](#).
- [130] Niels Justesen, Sebastian Risi, and Jean-Baptiste Mouret. “Map-elites for noisy domains by adaptive sampling”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2019, pp. 121–122 Cited on pages [28](#), [113](#).
- [131] Manon Flageat and Antoine Cully. “Fast and stable MAP-Elites in noisy domains using deep grids”. In: *Artificial Life Conference Proceedings*. MIT Press. 2020, pp. 273–282 Cited on pages [28](#), [113](#).
- [137] Timothée Lesort et al. “State representation learning for control: An overview”. In: *Neural Networks* 108 (2018), pp. 379–392 Cited on pages [32](#), [111](#).



- [141] Antoine Cully and Yiannis Demiris. “Hierarchical behavioral repertoires with unsupervised descriptors”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2018, pp. 69–76 *Cited on pages 33, 113.*
- [143] Charu Aggarwal, Alexander Hinneburg, and Daniel Keim. “On the Surprising Behavior of Distance Metric in High-Dimensional Space”. In: *First publ. in: Database theory, ICDT 200, 8th International Conference, London, UK, January 4 - 6, 2001 / Jan Van den Bussche ... (eds.). Berlin: Springer, 2001, pp. 420-434 (=Lecture notes in computer science ; 1973)* (Feb. 2002) *Cited on pages 35, 49, 53, 109.*
- [153] Austin Stone et al. “The Distracting Control Suite—A Challenging Benchmark for Reinforcement Learning from Pixels”. In: *arXiv preprint arXiv:2101.02722* (2021) *Cited on pages 41, 110.*
- [168] Rishabh Agarwal et al. “Contrastive behavioral similarity embeddings for generalization in reinforcement learning”. In: *arXiv preprint arXiv:2101.05265* (2021) *Cited on page 110.*
- [169] Cristian Bodnar et al. “A Metric Space Perspective on Self-Supervised Policy Adaptation”. In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4329–4336 *Cited on page 110.*
- [170] Xiang Fu et al. “Learning Task Informed Abstractions”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 3480–3491 *Cited on page 110.*
- [171] Huy Ha, Sian Lee Kitt, and William Zheng. “Deep Bisimulation Dreaming: Combating Distractions with State Abstractions”. In: () *Cited on page 110.*
- [172] Lucas Schott et al. “Improving Robustness of Deep Reinforcement Learning Agents: Environment Attacks based on Critic Networks”. In: *arXiv preprint arXiv:2104.03154* (2021) *Cited on page 110.*
- [173] Yuval Tassa et al. “Deepmind control suite”. In: *arXiv preprint arXiv:1801.00690* (2018) *Cited on page 110.*
- [174] Valentin Thomas et al. “Independently controllable factors”. In: *arXiv preprint arXiv:1708.01289* (2017) *Cited on page 111.*
- [175] Arun Pandey et al. “Disentangled Representation Learning and Generation with Manifold Optimization”. In: *arXiv preprint arXiv:2006.07046* (2020) *Cited on page 111.*
- [176] Francesco Locatello et al. “Challenging common assumptions in the unsupervised learning of disentangled representations”. In: *international conference on machine learning*. PMLR. 2019, pp. 4114–4124 *Cited on page 111.*



- [177] Mayalen Etcheverry, Clément Moulin-Frier, and Pierre-Yves Oudeyer. “Hierarchically Organized Latent Modules for Exploratory Search in Morphogenetic Systems”. In: *arXiv preprint arXiv:2007.01195* (2020) Cited on page 111.
- [178] Leo Cazenille. “Ensemble feature extraction for multi-container quality-diversity algorithms”. In: *arXiv preprint arXiv:2105.00682* (2021) Cited on page 111.
- [179] Yaochu Jin and Jürgen Branke. “Evolutionary optimization in uncertain environments—a survey”. In: *IEEE Transactions on evolutionary computation* 9.3 (2005), pp. 303–317 Cited on page 113.
- [180] Pratyusha Rakshit, Amit Konar, and Swagatam Das. “Noisy evolutionary optimization algorithms—a comprehensive survey”. In: *Swarm and Evolutionary Computation* 33 (2017), pp. 18–45 Cited on page 113.