Leveraging sequentiality in Robot Learning : Application of the Divide & Conquer paradigm to Neuro-Evolution and Deep Reinforcement Learning

Alexandre Chenu

Leveraging sequentiality in Robot Learning: Application of the Divide & Conquer paradigm to Neuro-Evolution and Deep Reinforcement Learning

ALEXANDRE CHENU

SORBONNE UNIVERSITÉ

Under the co-supervision of **Nicolas Perrin-Gilbert** and **Olivier Sigaud**.

In partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Board of examiners:

Jens Kober	Associate Professor	TU Delft	Reviewer
George Konidaris	Full Professor	Brown University	Reviewer
Aude Billard	Full Professor	EPFL	Examinator
Olivier Pietquin	Research Scientist	Google Brain	Examinator
Nicolas Perrin-Gilbert	Researcher	CNRS	Supervisor
Olivier Sigaud	Full Professor	Sorbonne Université	Director

Thesis defense date: 10/03/2023

Acknowledgements

The metaphor of a journey is often used to describe a PhD thesis. Although COVID did everything to limit mine to my desk and my sofa, I want to thank all the people without whom this 3-year journey would have been (way) shorter.

First and foremost, I want to thank my supervisors for their guidance, support, and encouragement throughout those three years. Your expertise, insights, and patience have been crucial in helping me complete this work. I can imagine how frustrating it was when I diverged for the umpteenth time toward a scientific mirage. Thank you for always having the patience to allow me to explore, to discuss my ideas (even the bad ones) and, above all, for always getting me back on track. I would have liked these three years to happen in simpler circumstances to benefit even more from our collaboration.

I want to thank my parents, for whom my love is infinite. Your benevolence has kept me going through the ups and downs. Thank you for putting up with my moods and stress during these three years and, above all, for believing in me even when I didn't. Thank you to my big brother for the same reasons but also for constantly pushing me to work on my curiosity. Thank you for being the model you have always been and will always be. Wherever they are, I also want to thank my grandparents for always supporting me in everything I have undertaken since I was a child, whether it was sports, music, or science. I wish you could have seen the result of these three years of work.

I want to thank all my friends, those I've always had, those from "prépa", Brest, London, from the lab. You can't imagine how valuable your support has been. Thank you for allowing me to return to the surface each time I was drowning in my work during these three years.

Thank you to my Ph.D. mates, my "Dean Moriarty". Those long days and evenings

spent in the lab would have been much less enjoyable without you. In particular, I want like to thank Olivier Serris for all the board sessions, constructive criticism and invaluable inputs. Thank you for transforming this thesis in only a few months of collaboration.

I would also like to thank the Open-Ended Learning research group, both for the short, off-topic rounds that punctuated Friday afternoons and for the fascinating discussions on this theme, which is as vast as its name implies.

I want to thank the Lycée Marie Curie, ENSTA Bretagne and Imperial College London for preparing me for this trip during the five years of study that preceded it. In particular, I want to thank Professor Antoine Cully for sharing his passion for research with me during the 6-month project that preceded this thesis.

I want to thank Sorbonne Université, the Ecole Doctorale Informatique Télécommunications et Electronique and especially the Institut des Systèmes Intelligents et de Robotique, for providing me with an excellent research infrastructure. Professor Cully advised me well, it is the perfect place for a Ph.D. thesis.

I want to thank the board of examinators for this thesis. I look forward to discussing with you at the Ph.D. defense.

I want to thank the Collège Des Ingénieurs for giving me the opportunity to follow their Science & Management program for 2 years. These regular meetings have been studious breaks in the thesis that have been as exciting as they will be valuable for the rest of my career.

Finally, I would like to thank the French National Research Agency (ANR) for financing this work through project ANR-18-CE33-0005 HUSKI.

Paris, 31/12/2022 A. C.

Abstract

"To succeed, planning alone is insufficient. One must improvise as well." This quote from Isaac Asimov, founding father of robotics and author of the *Three Laws of Robotics*, emphasizes the importance of being able to adapt and think on one's feet to achieve success. Although robots can nowadays resolve highly complex tasks, they still need to gain those crucial adaptability skills to be deployed on a larger scale.

Robot Learning uses learning algorithms to tackle this lack of adaptability and to enable robots to solve complex tasks autonomously. Two types of learning algorithms are particularly suitable for robots to learn controllers autonomously: Deep Reinforcement Learning and Neuro-Evolution. However, both classes of algorithms often cannot solve Hard Exploration Problems, that is problems with a long horizon and a sparse reward signal, unless they are guided in their learning process.
One can consider different approaches to tackle those problems. An option is to search for a diversity of behaviors rather than a specific one. The idea is that among this diversity, some behaviors will be able to solve the task. We call these algorithms Diversity Search algorithms. A second option consists in guiding the learning process using demonstrations provided by an expert. This is called Learning from Demonstration. However, searching for diverse behaviors or learning from demonstration can be inefficient in some contexts. Indeed, finding diverse behaviors can be tedious if the environment is complex. On the other hand, learning from demonstration can be very difficult if only one demonstration is available.

This thesis attempts to improve the effectiveness of Diversity Search and Learning from Demonstration when applied to Hard Exploration Problems. To do so, we assume that complex robotics behaviors can be decomposed into reaching simpler sub-goals. Based on this sequential bias, we try to improve the sample efficiency of Diversity Search and Learning from Demonstration algorithms by adopting Divide & Conquer strategies, which are well-known for their efficiency when the problem is composable.

Throughout the thesis, we propose two main strategies. First, after identifying some limitations of Diversity Search algorithms based on Neuro-Evolution, we propose Novelty Search Skill Chaining. This algorithm combines Diversity Search with Skill-Chaining to efficiently navigate maze environments that are difficult to explore for state-of-the-art Diversity Search.
In a second set of contributions, we propose the Divide & Conquer Imitation Learning algorithms. The key intuition behind those methods is to decompose the complex task of learning from a single demonstration into several simpler goal-reaching sub-tasks. DCIL-II, the most advanced variant, can learn walking behaviors for under-actuated humanoid robots with unprecedented efficiency.
Beyond underlining the effectiveness of the Divide & Conquer paradigm in Robot Learning, this work also highlights the difficulties that can arise when composing behaviors, even in elementary environments. One will inevitably have to address these difficulties before applying these algorithms directly to real robots. It may be necessary for the success of the next generations of robots, as outlined by Asimov.

Key words: *Diversity Search, Learning from Demonstration, Goal-Conditioned Reinforcement Learning, Robot Learning*

Résumé

"Pour réussir, il ne suffit pas de prévoir, il faut aussi savoir improviser." Cette citation d'Isaac Asimov, père fondateur de la robotique et auteur des Trois lois de la robotique, souligne toute l'importance d'être capable de s'adapter et d'agir dans l'instant présent pour réussir. Même si, aujourd'hui, les robots peuvent résoudre des tâches d'une complexité qui était inimaginable il y a encore quelques années, ces capacités d'adaptation leur font encore défaut, ce qui les empêche d'être déployé à une plus grande échelle.

Pour remédier à ce manque d'adaptabilité, les roboticiens utilisent des algorithmes d'apprentissage afin de permettre aux robots de résoudre des tâches complexes de manière autonome. Deux types d'algorithmes d'apprentissage sont particulièrement adaptés à l'apprentissage autonome de contrôleurs par les robots : l'apprentissage profond par renforcement et la neuro-évolution. Cependant, ces deux classes d'algorithmes ne sont capables de résoudre des problèmes d'exploration difficiles, c'est-à-dire des problèmes avec un horizon long et un signal de récompense rare, que s'ils sont guidés dans leur processus d'apprentissage. Différentes approches peuvent être envisagées pour permettre à un robot de résoudre un tel problème sans être guidé.
Une première approche consiste à rechercher une diversité de comportements plutôt qu'un comportement spécifique. L'idée étant que parmi cette diversité, certains comportements seront probablement capables de résoudre la tâche qui nous intéresse. Nous les appelons les algorithmes de recherche de diversité. Une deuxième approche consiste à guider le processus d'apprentissage en utilisant des démonstrations fournies par un expert. C'est ce qu'on appelle l'apprentissage par démonstration. Cependant, chercher des comportements divers ou apprendre par démonstration peut être inefficace dans certains contextes. En effet, la recherche de comportements divers peut être fastidieuse si l'environnement est complexe. D'autre part, l'apprentissage à partir d'une seule et unique démonstration peut être très difficile.

Dans cette thèse, nous tentons d'améliorer l'efficacité des approches de recherche par diversité et d'apprentissage à partir d'une seule démonstration dans des problèmes d'exploration difficiles. Pour ce faire, nous supposons que les comportements robo-

tiques complexes peuvent être décomposés en sous-comportements plus simples. Sur la base de ce biais séquentiel, nous adoptons une stratégie dite de "diviser-pour-régner", qui est bien connue pour être efficace lorsque le problème est composable.
Nous proposons deux approches en particulier. Premièrement, après avoir identifié certaines limites des algorithmes de recherche de diversité basés sur la l'évolution de réseaux de neurones artificiels, nous proposons *Novelty Search Skill Chaining*. Cet algorithme combine la recherche de diversité avec l'énchaînement de compétences pour naviguer efficacement dans des labyrinthes qui sont difficiles à explorer pour des algorithmes de l'état-de-l'art.
Dans une deuxième série de contributions, nous proposons les algorithmes *Divide & Conquer Imitation Learning*. L'intuition derrière ces méthodes est de décomposer la tâche complexe d'apprentissage à partir d'une seule démonstration en plusieurs sous-tâches plus simples consistant à atteindre des sous-buts successifs. DCIL-II, la variante la plus avancée, est capable d'apprendre des comportements de marche pour des robots humanoïdes sous-actionnés avec une efficacité sans précédent.
Au-delà de souligner l'efficacité du paradigme de diviser-pour-régner dans l'apprentissage des robots, cette thèse met également en évidence les difficultés qui peuvent survenir lorsqu'on compose de comportements, même dans des environnements élémentaires. Il faudra inévitablement résoudre ces difficultés avant d'appliquer ces algorithmes directement à des robots réels. C'est peut-être une condition nécessaire pour le succès des prochaines générations de robots, comme le souligne Asimov.

Mots clefs : *Recherche de diversité, Apprentissage par Demonstration, Apprentissage par Renforcement conditionné par des buts, Apprentissage des robots*

Acronyms

- **AGS** Approximated Goal Switching
- **BC** Behavioral Cloning
- **DB** Demo-Buffer
- **DCIL-I & DCIL-II** Divide & Conquer Imitation Learning I & II
- **DDPG** Deep Deterministic Policy Gradient
- **DoF** Degree-of-Freedom
- **DSC** Deep Skill Chaining
- **DRL** Deep Reinforcement Learning
- **DS** Diversity Search
- **EA** Evolutionary Algorithm
- **EST** Expansive Spaces Trees
- **GAN** Generative Adversarial Network
- **GEP** Goal Exploration Process
- **GC-MDP** Goal-Conditioned MDP
- **GCRL** Goal-Conditioned Reinforcement Learning
- **GGI** Goal-Guided Imitation
- **HEP** Hard Exploration Problem
- **HER** Hindsight Experience Replay

- **HRL** Hierarchical Reinforcement Learning
- **IL** Imitation Learning
- **IM** Intrinsic Motivation
- **IRL** Inverse Reinforcement Learning
- **LfD** Learning from Demonstration
- **MDP** Markov Decision Process
- **MLP** Multi-Layer Perceptron
- **MP** Motion Planning
- **MSE** Mean Squared Error
- **MSBE** Mean Squared Bellman Error
- **NE** Neuro-Evolution
- **NS** Novelty Search
- **NSSC** Novelty Search Skill Chaining
- **PWIL** Primal Wasserstein Imitation Learning
- **RB** Replay-Buffer
- **R-DSC** Robust Deep Skill Chaining
- **RL** Reinforcement Learning
- **RLfD** Reinforcement Learning from Demonstration
- **RRT** Rapidly-exploring Random Tree
- **SAC** Soft-Actor Critic
- **SACBC** Soft-Actor Critic Behavioral Cloning
- **SACfD** Soft-Actor Critic from Demonstration
- **SACR2** Soft-Actor Critic Reward Relabelling
- **SR-DCIL** Single-Reset Divide & Conquer Imitation Learning
- **TD** Temporal-Difference

- **TD3** Twin Delayed Deep Deterministic Policy Gradient
- **TQC** Truncated Quantile Critics
- **VC** Value-Cloning

Contents

1 Introduction

There are examples of mechanical machines performing simple tasks autonomously dating back centuries, but the concept of robots has long remained almost absent from the collective imagination. In the early twentieth century, truly autonomous robots began to appear in literature and comic books (Baum, 1907; Collier, 1911), with capabilities far exceeding those of their physical world counterparts. The term robot was soon coined (Capek, 1921), and fictional robots became increasingly popular in the following decades (Asimov, 1950). At the same time, the birth of electronics led to more impressive autonomous machines, such as Elektro in 1939 (Marsh, 2018), and robotics gradually became a research field in the 1950s and 1960s, with precursory telerobotics and industrial robotics (Mason, 2012).

However, the capabilities of physical robots remained far from the unlimited possibilities of fiction, which generated very high expectations. Until the beginning of the present century, the most common robots were repeating the same standardized task over and over again in the fully controlled environment of their factory. Advances in robotics have been steady, but have arguably not kept pace with advances in computer power, and despite significant improvements in the complexity of tasks that robots can solve, we are still a long way from robots with human-like capabilities.

At the 2015 DARPA challenge, the numerous crashes[I] highlighted that robots were not ready to be deployed in complex and unknown environments (Atkeson et al., 2015). More intriguingly, the robots showcased as useful for exploration in extreme environments (Krotkov et al., 2018; Tranzatto et al., 2022) have had limited applicability in a real-world context such as the Fukushima disaster (Tsitsimpelis et al., 2019). These failures highlighted the limits of traditional robotics combining mechanical modeling and optimization. They led to intensive research focused on the adaptability and

[I] https://www.bbc.com/news/av/technology-33049253

robustness of robots in non-laboratory conditions, resulting in a new generation of impressive robots. For instance, at the recent DARPA's Subterranean Challenge 2021, the winning robot, a quadruped ANYmal (Hutter et al., 2017) from team Cerberus (ETH Zurich) (Tranzatto et al., 2022), was able to map and locate itself in a network of underground tunnels while walking on rough terrain. Another example of these advances is the recent achievement of the humanoid robot Atlas from Boston Dynamics doing parkour [II].

However, although very efficient at solving these complex tasks, these robots are still far from the robots with a wide range of skills and near-human adaptability imagined in science fiction novels and movies. Indeed, despite significant advances, the lack of adaptability of robots remains a major limitation. For instance, minimal changes resulting from damages (Cully et al., 2015) or a varying coefficient of friction of the floor (Kajita et al., 2004) are enough to prevent a robot from completing a task. To tackle this lack of adaptability, a growing part of the robotics community is turning to learning algorithms to avoid the burden of characterizing the infinite number of highly complex tasks and situations a robot might face.

Learning algorithms aim to endow an artificial agent with the capability to acquire knowledge and skills to solve a task without being explicitly programmed to do so (Mitchell, 1997). Using such algorithms in robots has given rise to the field of *Robot Learning* (Connell and Mahadevan, 2012; Peters et al., 2016). Among the different categories of learning algorithms, *Reinforcement Learning* (RL) (Sutton and Barto, 1998), which consists in optimizing sequential decision-making, is very well adapted to the context of robot control (Kober et al., 2013). On the other hand, *Evolutionary Algorithms* (EAs) are also interesting because of their *black-box* nature and the few assumptions required to use them (Bongard, 2013; Doncieux et al., 2015). When these algorithms are applied to learn a control policy corresponding to an artificial neural network (Jain et al., 1996), they are called Deep Reinforcement Learning (DRL) (Mnih et al., 2013) and (Deep) Neuro-Evolution (NE) (Stanley and Miikkulainen, 2002) respectively.

Most learning algorithms are based on optimizing an objective function (Alpaydin, 2020). The simplest class of learning algorithms is *Supervised Learning* (Hastie et al., 2009). In a Supervised Learning task, a teacher provides the artificial agent with error feedback proportional to the difference between the system estimate and the desired value. For example, in image classification, the objective is to minimize the error between the image class estimate and its true class. RL and EA methods use a slightly more complex type of objective function. In RL, it is assumed that there is a function

[II] https://www.bostondynamics.com/resources/blog/leaps-bounds-and-backflips

that gives quantitative feedback on the actions that an artificial agent has performed. This function is called the reward function (Sutton and Barto, 1998). Thus, a robot learns by maximizing its reward over time. In EAs, an analogous function called the fitness function is assumed to exist (Eiben, J. E. Smith, et al., 2003) and assesses the agent's performance.

Given the recent successes of Robot Learning methods (Cully et al., 2015; Andrychowicz et al., 2017; Sermanet et al., 2018; Akkaya et al., 2019; Ecoffet et al., 2021), it is tempting to call upon these methods to solve minimally specified tasks and circumvent any kind of complex behavior specification. However, the less specified the objective function is, the more difficult it is for an agent to learn. One could use a minimal task specification to define the objective function. However, this likely results in a sparse reward problem. For instance, let's consider a humanoid robot learning how to stand up using an RL algorithm. If one defines the reward function as positive when the chest of the robot is above a certain height and null elsewhere, the reward is sparse. Without a dense reward signal, the agent cannot benefit from any learning curriculum and will probably learn no interesting behavior. On the other hand, a dense but poorly defined reward causes the agent to learn a sub-optimal behavior that is unfit to solve the task. This is called a deceptive reward. In the standup task, naive dense feedback could correspond to a reward function proportional to the distance between the chest and the ground. However, such a reward function gives no information about the complex coordination of hands and feet that a humanoid needs to have in order to stand up. At best, using such a reward may result in a sub-optimal behavior in which the robot sits down rather than standing up. Whether the reward is sparse or deceptive, the problem is the same: in such scenarios, the artificial agent cannot learn by naively optimizing its objective function.

Learning from such ill-defined reward functions falls into the class of *Hard-Exploration Problems* (HEP). More precisely, an HEP is an RL problem where the reward function is either sparse or deceptive, a long control sequence is required to solve the task, and the environment dynamics are highly non-linear or even discontinuous. A popular toy Hard-Exploration Problem is a 2D maze navigation task where the agent has no perception of the walls (Matheron et al., 2020; Pitis et al., 2020; Castanet et al., 2022).

To solve HEPs, several *Robot Learning* communities defined alternative objective functions to solve the task indirectly. The *Open-Ended Learning* community (Doncieux et al., 2018) draws on biology and developmental psychology to define objective functions that encourage an artificial agent to find a diversity of behaviors rather than one particular behavior. The idea is that within this diversity of behaviors, certain behaviors will be able to solve the task. The work of this community is based on the

intrinsic motivation of infants and animals (Berlyne, 1950; Berlyne, 1966; Gopnik et al., 2001; Gottlieb and Oudeyer, 2018) and includes algorithms such as *Diversity Search Evolutionary Algorithms* (Lehman and Stanley, 2011a; Forestier and Oudeyer, 2016; Cully and Demiris, 2018b) or *Intrinsically Motivated Reinforcement Learning* algorithms (Schmidhuber, 1991; Colas et al., 2022). In the example of the humanoid trying to stand up, these approaches seek to move in as many different ways as possible to eventually find a behavior that results in the robot getting up.

A second approach to tackle Hard Exploration Problems is *Learning from demonstration* (Schaal, 1996; Schaal, 1999). Here, the task to be solved is specified by one or more demonstrations and the agent seeks to reproduce the demonstrated behavior to complete the task (Ravichandar et al., 2020). This is the approach we leverage in this work.

1.1 Motivation

Both *Diversity Search* (DS) and *Learning from Demonstration* (LfD) have been successfully applied to learn controllers for complex robots in minimal specification settings (Calinon and Billard, 2007; Kober and Peters, 2011; Cully et al., 2015; Ho and Ermon, 2016b; Such et al., 2017; Dadashi et al., 2020). However, they face several limitations in this precise context preventing them from being widely applied.

Firstly, the more complex the robot, the more interactions DS approaches require to learn diverse behaviors. Indeed, DS algorithms often use a low-dimensional space to characterize the behavior of controllers and to guide the search for diversity. However, exploring this low-dimensional space may be difficult when the robot is complex. For instance, if the robot is subject to complex non-linear dynamics, most controllers will probably generate similar and uninteresting behavior. Therefore, finding rare controllers yielding interesting behaviors can be very challenging.

Despite avoiding a potentially complex search for diverse behaviors, costly learning also arises in LfD as in general many interactions with the robot are required to learn the controller (Arora and Doshi, 2021). This latter issue is particularly important since collecting a high number of demonstrations can be challenging in certain contexts (Stadie et al., 2017). Because of the high cost of these learning methods, their direct application to physical robots in real-world contexts is still limited. Therefore, it is essential to tackle these limitations to improve their sample efficiency.

1.1.1 Research Question

In robotics, a common approach to simplify complex problems is to exploit a sequential bias. For instance, one can assume that robotic movements are sequential and composable. Indeed, a complex movement can often be decomposed into a sequence of simpler movements. Based on this assumption, different robotics communities ranging from Robot Learning (Kober et al., 2015; Yan et al., 2020; Allard et al., 2022) to Optimal Control (Bock and Plitt, 1984; Diehl et al., 2006) have adopted Divide & Conquer strategies (Knuth, 1997), known to be efficient when a problem is compositional, to decompose a complex controller into several simpler controllers.

In this work, we wonder under which conditions we can exploit a sequential bias in robotics tasks to improve the efficiency of DS and LfD algorithms when tackling a specific category of HEPs corresponding to continuous control problems in simulated robotic environments without any exploitable objective function. To investigate the question, we adopt a Divide & Conquer strategy and we show that learning a sequence of skills into a unique neural network conditioned on a sequence of goals raises specific issues when the system to be controlled is of higher dimension than the goals.

In more details, we focus mainly on environments with strong non-holonomic constraints because many difficulties arise when trying to compose successive behaviors in the presence of such constraints. In particular, because of these constraints, it can be difficult to successively reach two goals that are apparently very close. Therefore, when composing two behaviors, special care must be taken to ensure that a sub-behavior ends in precise states that allow the following sub-behavior to be completed. This corresponds to the notion of *valid states*, which is crucial throughout this thesis.

1.2 Outline & contributions

The first part of the document sets out the key concepts used throughout this thesis. Chapter 2 introduces fundamental principles in robotics. It starts by presenting a generic characterization of the state of a robot based on the concept of configuration. All the experiments presented in this thesis rely on this characterization. With this definition in mind, Motion Planning problems are defined and sampling-based Motion Planning algorithms are introduced. In chapters 3 and 4, an overview of RL and NE is given. These successive presentations of RL and NE focus on the problem of learning from poorly defined reward functions in HEPs. After explaining the problem, several alternatives are proposed, such as *Intrinsic Motivation*, *Diversity Search*, etc. The final chapter of this part introduces the various ways of learning from demonstrations, a supplementary alternative to tackle HEPs when demonstrations are available.

The second part of the document progressively presents all the contributions of this thesis. In Chapter 6, we assess the ability of DS algorithms to learn control policies to solve HEPs. In other words, we want to learn diverse policies to solve a complex robotics task without access to any informative fitness or reward function. By drawing a parallel with sampling-based Motion Planning algorithms, we highlight the poor sample efficiency of DS algorithms when applied to HEPs involving continuous control. Indeed, when the environment dynamics are complex, the fragility of the mutation operators used in NE makes the search for diverse behaviors very inefficient. Adopting a Divide & Conquer strategy, in Chapter 7, we propose to use the concept of skill-chaining to transform a difficult search for diverse complex behaviors into a simpler search for locally diverse sub-behaviors. Although effective in a certain class of HEPs, the proposed approach presents critical limitations in non-holonomic environments. Indeed, searching for a locally diverse behavior may be inefficient if the behavior starts from a state that is strongly constrained. We call these states *invalid exploration states*.

In Chapter 8, we propose to tackle the problem of learning complex robotics policies with LfD. To this end, we propose Divide & Conquer Imitation Learning I (DCIL-I), an approach based on Goal-Conditioned Reinforcement Learning designed to learn a complex behavior from a single demonstration. Once again, DCIL-I adopts a Divide & Conquer strategy and decomposes a difficult imitation task into several simpler goal-reaching subtasks. Given the difficulty of achieving a particular high-dimensional desired state, DCIL-I proposes to achieve high-level low-dimensional goals (e.g. successive positions of the torso of a humanoid in a locomotion task). However, high-level goals do not condition the actual state of the agent when it achieves a goal, which can be very problematic, particularly in non-holonomic environments. Therefore, DCIL-I proposes mechanisms to encourage the agent to always achieve a goal by reaching states compatible with the future achievement of the next goal. We call those compatible states *valid success states*. DCIL-I was successfully applied to solve a navigation task and a grasping task from a single demonstration.

Chapter 9 presents DCIL-II, a variant of DCIL-I based on an original extended Goal Conditioned RL framework. DCIL-II benefits from improved sample efficiency and was successfully applied to complex locomotion tasks. Finally, in Chapter 10, several variants of DCIL are proposed to remove the limiting assumption in DCIL-I and DCIL-II that the agent may be reset to any demonstration state.

In summary, the contribution of this thesis are the following:

- We identified some limitations of the mutation operator of NE algorithms when

applied to DS to tackle HEP in environments with complex dynamics (Chenu et al., 2021).

- We proposed Novelty Search Skill-Chaining, a novel approach to overcome the identified limitations based on the idea of dividing a complex search for diverse behaviors into several simpler searches for locally diverse behaviors. While greatly improving the sample efficiency of DS in a certain class of environments, limits arise in non-holonomic environments.

- We proposed DCIL-I (Chenu, Perrin-Gilbert, and Sigaud, 2022), a Goal Conditioned RL approach dividing the imitation of a complex behavior specified by a single demonstration into simpler goal-reaching subtasks. DCIL-I is specifically designed to learn a goal-conditioned control policy to sequentially reach high-level low-dimensional goals in a complex high-dimensional continuous state space.

- We proposed DCIL-II (Chenu, Serris, et al., 2022) a more sample-efficient variant of DCIL-I based on an original Goal Conditioned-RL framework that makes it possible to deal with a sequence of goals with a goal-conditioned policy.

- We proposed SR-DCIL, an extension of DCIL-II that weakens the state resetting assumption required by DCIL-I and DCIL-II.

Background Part I

In brief

This first part of the thesis introduces the fundamental concepts on which the contributions are built. The first chapter gives a brief introduction to fundamental concepts in robotics. It starts by presenting a generic state representation of an articulated robot. Then, Motion Planning problems and algorithms designed to solve them are presented. The second and third chapters give a non-extensive overview of the field of Reinforcement Learning and Neuro-evolution. Those presentations are structured around the limits of Robot Learning in Hard Exploration Problems, i.e. in the absence of informative training signals like well-defined reward functions. Several approaches based on Reinforcement Learning and Neuro-Evolution and designed to tackle Hard Exploration Problems are presented in both chapters. Finally, Chapter 5 introduces the alternative solution of learning from demonstration.

2 Background in robotics

2.1 Representing the state of a robot

The term state has different meanings in physics depending on the system studied. In thermodynamics, a state specifies the physical properties of a system (e.g., temperature, volume, pressure, etc.) (Cengel et al., 2011). In quantum physics, the quantum state is a probability distribution over the possible outcomes of a measurement (Griffiths and Schroeter, 2018). However, each of these definitions answers the fundamental question: "what is the condition of the system of interest?"

This question, when raised in the context of robotics, can be divided into two parts. First, one must specify the position of a robot. This involves the notion of a configuration, a parameterized representation of the position of all points of the robot. Second, by augmenting configurations with velocities, one can characterize the motion of the robot and obtain states of a robot or an environment.

2.1.1 Representing the position of a robot

To answer the first sub-question: "what is the position of my robot?", one must start by characterizing the robot. A robot can be described by a set of rigid bodies called links. While more complex robot representations are used in soft robotics (Laschi et al., 2017), all the robots considered in this thesis can be modeled as rigid bodies. These links are connected together via two types of joints. If a force or a torque can be applied to a joint, the joint is *actuated*. If not, the joint is *unactuated*.

The *configuration* of a robot offers a parameterized representation of the position of all points of the robot.

Since a link is rigid, all its points are attached to an associated coordinate system (see Figure 2.1). This coordinate system is described by 3 degrees of freedom (DoF) for its position and 3 DoF for its orientation. Therefore, the configuration of a link is given by 6 parameters.

By extrapolation, a naive representation of the configuration of a robot with N links is given by $6N$ parameters corresponding to the 6 DoF of all links. However, constraints can be applied to the robot, reducing the number of DoF. Each *independent* constraint removes one DoF from the robot. The number of DoF of a robot can be calculated using the **Chebychev-Grübler-Kutzbach criterion** (Uicker et al., 2003).

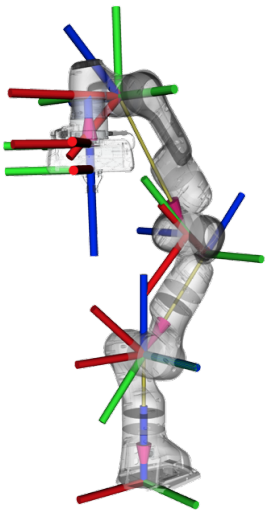


Figure 2.1: Robot arm example. This robot can be described as 9 rigid links. All the points of a link are attached to a coordinate system represented by 3 axes (red, green and blue arrows). Figure taken from http://docs.ros.org/en/kinetic/api/moveit_tutorials/html/.

Theorem 1 *(Uicker et al., 2003) Consider a robot with N links, where the ground is also regarded as a link. Let J be the number of joints, m be the number of DoF of a rigid body ($m = 3$ for planar systems and $m = 6$ for spatial systems), f_i be the number of freedoms provided by joint i and c_i be the number of constraints provided by joint i ($f_i + c_i = m$ for all i). Then the number of DoF N_{DoF} of the robot is :*

$$\begin{aligned} N_{DoF} &= m(N-1) - \sum_{i=1}^{J} c_i \\ &= m(N-1) - \sum_{i=1}^{J} (m - f_i) \\ &= m(N-1-J) + \sum_{i=1}^{J} f_i \end{aligned} \tag{2.1}$$

Therefore, by taking these constraints into account, the configuration of a robot with N links with constraints can be expressed using less than $6N$ parameters. Using such minimal parametrization, the set of configurations of a robot with N links corresponds to a N_{DoF} manifold embedded in a $6N$-dimensional space called the configuration space and noted C. For instance, the configuration space of a double pendulum corresponds to a 2D torus embedded in $\mathbf{R}^3$ (see Figure 2.2).

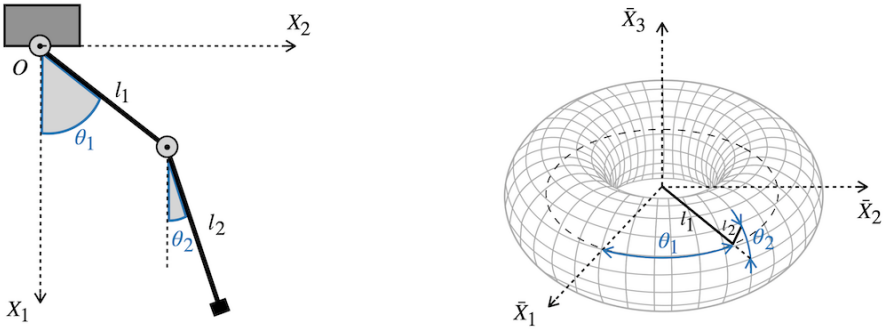


Figure 2.2: An example of a double pendulum system and its configuration space (a 2D torus embedded in $\mathbb{R}^3$. Figure adapted from (Awrejcewicz, 2012).

2.1.2 Representing the motion of a robot

The configuration provides information about where the robot is at a given time. However, robots are dynamical systems. The forces or the torques applied on its links (e.g. inertia or gravity) make the robot move over time. To fully describe the motion of a robot, in addition to positional information, one should take into consideration the velocity $\dot{q}$ of each DoF of the robot. Therefore, the state s of a robot can be written as a couple $(q, \dot{q})$ where q corresponds to the configuration of the robot.

In a complex environment involving several robots or objects, assuming perfect knowledge of the position and velocities of each environment element, the richest state representation corresponds to the concatenation of the state of all elements. In the robotics environment of this thesis, we assume that such knowledge about the environment is available.

2.2 Background on Motion Planning

One can resolve a robotic task in certain contexts without learning a control policy. Suppose the task is to achieve a desired configuration of the robot. In addition, suppose that the environment is not too noisy or that a low-level close-loop controller (e.g. Proportional Derivative or Linear Quadratic Regulator controllers (Jeff B Burl and Jeffrey B Burl, 1999)) is available to correct noise-related deviations. Then, finding a sequence of commands to apply to the system (or a sequence of configurations to reach) is sufficient to achieve the desired configuration reliably. One can call upon motion planning algorithms (LaValle, 2006; Latombe, 2012) to find such a sequence of controls or configurations and, thus, convert a human-level task into a sequence of movements that satisfies it.

This Section formally defines a Motion Planning problem, introduces sampling-based motion-planning algorithms, and defines the critical concept of non-holonomic constraints.

2.2.1 Motion Planning problem definition

In Motion Planning (MP), the goal is to find the appropriate commands to control a system from a starting configuration to a goal configuration or region. More formally, an MP problem can be defined by a (C, q_0, C_{target}) triplet where:

- C is the space of all possible configurations of the robot. It contains C_{free}, the subspace of free configurations of the robot. C_{free} excludes the configurations causing collisions with obstacles C_{obs} i.e. $C_{free} = C \setminus C_{obs}$.
- q_0 is a starting configuration of the robot.
- C_{target} is the space of target configurations.

The goal is to find a valid trajectory $\tau = (q_i)_{i=0,\dots,n}$ between q_0 and C_{target}. Depending on whether a model of the robot and the position of the obstacles are known, exploring C_{free} to find a valid trajectory has varying difficulty.

2.2.2 Sampling-based Motion Planning algorithms

Sampling-based MP algorithms rely on the construction of an exploration tree to construct paths in C_{free} (LaValle, 2006). The exploring tree is constructed from the starting configuration of the robot. Nodes are configurations and edges represent the fact that the system can navigate between the two linked configurations. When the model of the system is known, one can rely on a controller to find a sequence of controls to navigate between two nodes. Therefore, two successive nodes can potentially be far away from each other. However, in a minimal assumption setting where no model of the system is available, predicting the effects of a command is difficult. In that case, one must call upon random controls for a few time steps and rely on the fact that interesting motions may occur if many random actions are tried.

Thus, a sampling-based MP algorithm can be summarized as a two-step loop. First, a node is selected according to a selection operator. Then, the tree is expanded from the selected node. If a controller is available, it may direct the expansion toward a precise configuration. Otherwise, the tree is expanded using random commands.

Different selection operators have been proposed in the MP literature. Rapidly-exploring Random Trees (RRT) (LaValle et al., 1998) randomly draws a sample in the search space and selects the closest node in the exploration tree. Instead, Expansive Spaces Trees (EST) (David Hsu et al., 1998) selects nodes that lie in a region of low node density of the exploration tree. This difference in selection strategy results in RRT and EST having different exploration dynamics. These selection strategies are detailed in Chapter 6.

2.2.3 Non-holonomic constraints

Equality constraints exclusively involving the configuration $q = \{q_i\}_{i \in N_{DoF}}$ of the systems are called *holonomic*. More complex constraints (e.g. involving the derivative of the configuration) that cannot be written as such equalities are called *non-holonomic* (Tanedo, 2013):

$$\begin{aligned} &f(q_1, q_2, ..., q_{N_{DoF}}, t) = 0 \ \textit{holonomic} \\ &f(q_1, q_2, ..., q_{N_{DoF}}, \dot{q_1}, \dot{q_2}, ..., \dot{q_{N_{DoF}}}, t) = 0 \ \textit{non-holonomic} \\ &f(q_1, q_2, ..., q_{N_{DoF}}, t) < 0 \ \textit{also non-holonomic} \end{aligned} \tag{2.2}$$

For example, the distance between two joints of a robot connected by a link cannot change. This is a holonomic constraint. Certain holonomic constraints can also be described on the derivative of the configuration $\dot{q}$. However, as they are *integrable*, they can be integrated to recover a constraint on the configuration. A system only subject to holonomic constraints is called a *holonomic* system. On the other hand, constraints that cannot be integrated as equality constraints on configurations are non-holonomic. A system with at least one non-holonomic constraint is *non-holonomic*.

For example, the Reeds-Shepp Car is a non-holonomic system (reeds and Shepp, 1990) which includes a non-holonomic constraint that prevents the systems from sliding sideways. The following equations define its motion:

$$\begin{cases} \dot{x} = u_1 \cos(\theta) \\ \dot{y} = u_1 \sin(\theta) \\ \dot{\theta} = u_1 u_2 \end{cases} \tag{2.3}$$

where $u_1 \in \{-1, 1\}$ controls the forward/backward motion and u_2 controls the orientation.

In a holonomic problem, the agent can achieve all trajectories respecting the constraints on the configuration (defined as (2.2)). Therefore, any trajectory on the configuration manifold is admissible. Moreover, trajectories can be expressed directly in the configuration space as configuration changes are independent of the velocities. This can be seen as planning a path rather than a motion.

However, robotics systems are often non-holonomic. Indeed, the dynamics of the robot and the gravity induce non-holonomic constraints. In this context, "close" configurations may require an indirect trajectory in the configuration space to be connected. For instance, the shortest path with a non-holonomic Reeds-Shepp Car robot differs from the shortest path obtained if the car is allowed to slide (see Figure 2.3).

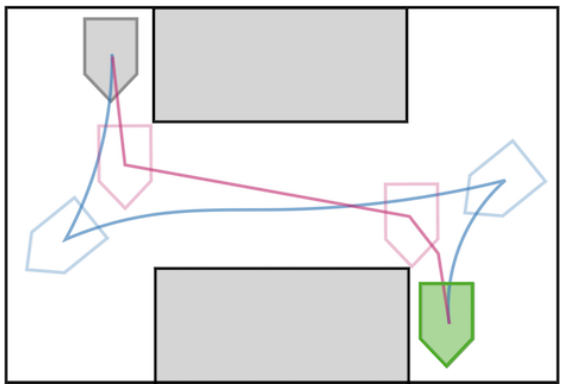


Figure 2.3: Example of the shortest path for a Reeds-Sheep Car (in blue) between two configurations (from grey to green) and for an imaginary "car" allowed to slide (in purple).

2.3 Summary

One can obtain a simple representation of a robot using rigid bodies and joints. In addition, considering the constraints applied to the joint of the robot results in a minimal specification of its position. To fully characterize the motion of a robot, the configuration must be extended with the derivative of its elements, which results in rich representation of the robot state. In a complex environment composed of more than a single robot, the environment state is obtained by aggregating the individual state of all elements.

In an MP problem, one searches for a valid trajectory in a robot's configuration space between a starting configuration and a set of desired ones. One can call upon sampling-based MP algorithms to resolve an MP problem. A sampling-based MP algorithm constructs an exploring tree from the starting configuration to find a solu-

tion path for the MP problem. However, non-holonomic constraints can make an MP problem difficult to solve, as connecting two close configurations may not be trivial.

3 Background on Reinforcement Learning

Reinforcement Learning has been an active research field for the past four decades leading to impressive results in the context of Robot Learning (Kober et al., 2013; Arulkumaran et al., 2017; Akkaya et al., 2019; Tsounis et al., 2020). While traditional RL algorithms were designed to work with small and discrete state and actions spaces, their recent combination with Deep Learning techniques (Goodfellow et al., 2016) opened up the path toward Deep Reinforcement Learning (DRL) algorithms. These algorithms have been successfully applied to problems with complex continuous and high-dimensional state and action spaces. A recent impressive result from the Deep Reinforcement Learning community is the application of the Dreamer algorithm to learn a robust locomotion behavior with a physical quadruped (P. Wu et al., 2022). Dreamer learned a policy that enables a quadruped robot to stand up and walk solely by interacting for $\sim$ 1 hour with the physical system. To put this result into perspective, one should note that newborn foals, which are among the fastest mammal to learn walking behaviors, take approximately the same amount of time to learn how to stand and walk (40 minutes to 1 hour) (Curcio and Nogueira, 2012).

This chapter introduces the frameworks, algorithms and concepts on which the Reinforcement Learning parts of this thesis are built. First, the usual framework for Reinforcement Learning (RL) problems is presented. Based on this framework, traditional RL algorithms such as Value Iteration (Bellman, 1957), Temporal Difference learning (Sutton, 1988) and Q-learning (Watkins and Dayan, 1992) algorithms are introduced and used to derive the Soft Actor-Critic algorithm (Haarnoja, Zhou, Abbeel, et al., 2018), a state-of-the-art Deep RL algorithm used throughout this thesis. In addition, the importance of the reward function is discussed and different mechanisms allowing RL to tackle Hard Exploration Problems are presented.

Then, the RL framework is extended to Goal-Conditioned RL (GCRL). Several mechanisms designed to help GCRL cope with sparse reward are also presented.

Finally, based on the previous Chapter 2, the application of sampling-based MP algorithms to solve RL problems is discussed. The main purpose of this discussion is to underline the limit arising when using such algorithms in non-holonomic environments with high dimensional states. Indeed, analogous limits arise in the application of the Divide & Conquer strategy in Chapters 7,8,9 and 10.

3.1 Reinforcement Learning framework

3.1.1 Markov Decision Processes

To formalize Reinforcement Learning (RL) problems, one generally calls upon Markov Decision Processes (MDP) (Bellman, 1966; Sutton and Barto, 1998). An MDP is defined as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, p, \gamma)$ where

- $\mathcal{S}$ is the set of possible *environment states*. In a robotic environment, a state may correspond to the definition given in Section 2.1.2. However, states may have different meanings and even be discrete (e.g. the positions of each piece in a board game (Silver et al., 2016; Silver et al., 2017; Silver et al., 2018)).
- $\mathcal{A}$ is the set of possible *actions*. In robotics, actions correspond to the control applied to modify the state of the system. It may correspond to the torques applied to the controllable joints. If a low-level controller (e.g. a Proportional Derivative controller) is available, actions may also correspond to desired positions (X. B. Peng et al., 2018). Similarly to states, actions can be continuous or discrete (Mnih et al., 2013; Salimans and R. Chen, 2018; Ecoffet et al., 2021).
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbf{R}$ is the unknown *reward function*. The reward function maps the triplet (state, action, and next state) to a feedback signal used to train the agent. For instance, in a robotic arm environment, a simple reward function may be the distance between the configuration q of the robot (contained in s) and the desired one q_{goal} (i.e. $R(s, a, s') = |q' - q_{goal}|_2$).
- $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the unknown *transition probability function*. It returns the probability of the environment transitioning from state s to state s' after applying action a. In a deterministic environment, $p(s'|a, s) = 1$ for any triplet $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$. In robotics, the stochastic nature of the transitions usually comes from non-modeled physical noise or uncertainty in the sensors.
- $\gamma \in [0, 1]$ is the *discount factor*. It characterizes the importance of long-term rewards in a sequence of interactions (Sutton and Barto, 1998).

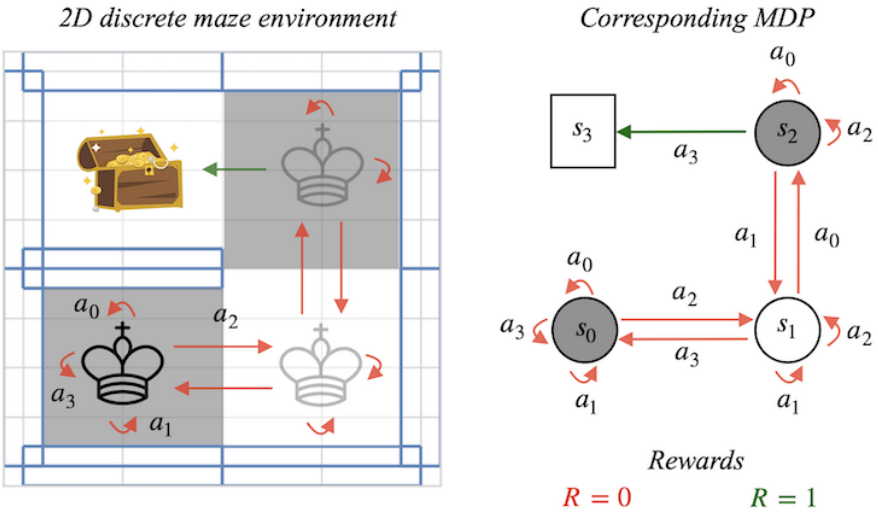


Figure 3.1: *Simple 2D discrete and deterministic maze environment and its corresponding MDP.* The four states $S = \{s_0, s_1, s_2, s_3\}$ correspond to the four colored squares of the maze. In each state, the agent (represented by a chess piece) can choose between four actions: {up, down, right, left}. When a collision with a wall occurs, the agent remains in the same state. The reward received by the agent is zero in every state except s_3 which corresponds to the treasure. Therefore, the reward is *sparse*.

Figure 3.1 presents a simple 2D environment with its corresponding MDP. The state space $\mathcal{S} = \{s_0, s_1, s_2, s_3\}$ and the action space $\mathcal{A} = \{a_0, a_1, a_2, a_3\}$ are discrete. In addition, transitions are deterministic.

In such an environment, at each step of a time sequence $t = 0, 1, 2, ..., T$, the agent interacts with the environment. It selects an action $a_t \in \mathcal{A}$ based on its current state $s_t \in \mathcal{S}$ using a policy $\pi(a_t|s_t)$. π which can have different forms. It can either be stochastic or deterministic. If π is stochastic, it maps a state to a distribution over actions. If π is deterministic, it directly maps a state to a unique action. Moreover, it can correspond to a non-parametric or a parametric function. For instance, in Deep Reinforcement Learning, policies correspond to neural networks.

After applying the selected action, the agent moves to a new state s_{t+1} according to $p(s_{t+1}|s_t, a_t)$ and receives a reward $R(s_t, a_t, s_{t+1})$. In an MDP, given current state s_t and action a_t, the next state does not depend on the previous states $(s_{t-1}, s_{t-2}, ...)$. This is a key assumption called the *Markov Assumption* which greatly simplifies RL problems and is made by most RL algorithms (Sutton and Barto, 1998).

The objective in RL is to obtain a policy that maximizes the expected discounted cumulative reward $J(\pi)$ defined as

$$J(\pi) = \mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t, S_{t+1})\right], \tag{3.1}$$

where S_t and A_t denote the random variables modeling the state visited and the action taken at time step t. Note that the initial state S_0 is drawn according to a known or unknown initial probability distribution ρ_0.

The discount factor diminishes the importance of long-term rewards. It accounts for the fact that long-term rewards are less desirable than short-term rewards. For instance, if one is hungry now, they would rather have a meal sooner than later. Practically, γ also ensures that the sum (3.1) converges to a finite return in the context of infinite time-horizon and in the presence of infinite loops in the state transitions.

3.1.2 Value and Q-value functions

Two central tools are used by almost all RL algorithms to learn the policy that maximizes (3.1). Those tools are the *Value function* and the *Q-value function*.

Given a state s, the *Value function* corresponds to the expected discounted return when following policy π from state s:

$$V_\pi(s) = \mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t, S_{t+1}) | S_0 = s\right]. \tag{3.2}$$

Similarly, given a state s and an action a, the *Q-value* is defined as the expected return when following policy π after taking action a in state s:

$$Q_\pi(s, a) = \mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t, S_{t+1}) | S_0 = s, A_0 = a\right]. \tag{3.3}$$

Roughly speaking, the *Value* and *Q-value functions* estimate *"how good"* it is for an agent following policy π to respectively be in state s and apply action a in state s.

The *Bellman Equations* decompose V_π and Q_π into recursive relationships between the immediate reward and the discounted future value:

$$V_\pi(s) = \mathop{\mathbb{E}}_{\substack{A\sim\pi \\ S'\sim p}} \left[R(S,A,S') + \gamma V_\pi(S') | S = s \right], \tag{3.4}$$

$$Q_\pi(s,a) = \mathop{\mathbb{E}}_{S'\sim p} \left[R(S,A,S') + \gamma V_\pi(S') | S = s, A = a \right], \tag{3.5}$$

Bellman Equations

where S, A and S' are the random variables modeling the current state, the action taken and the next state, respectively.

These recursive relationships are the cornerstone of the algorithms presented in the following paragraphs.

3.2 Paving the way to Soft-Actor-Critic

In an MDP with discrete states and actions (e.g. representing a board game), the expectation in *Bellman Equations* (3.4) (3.5) can be decomposed as an explicit sum over states and actions:

$$\begin{aligned} V_\pi(s) &= \mathop{\mathbb{E}}_{\substack{A\sim\pi \\ S'\sim p}} \Big[R(S,A,S') + \gamma V_\pi(S') | S = s \Big], \\ &= \sum_{a\in\mathcal{A}} \pi(a|s) Q_\pi(s,a), \\ &= \sum_{a\in\mathcal{A}} \pi(a|s) \underbrace{\sum_{s'\in\mathcal{S}} p(s'|s,a) \Big(R(s,a,s') + \gamma V_\pi(s') \Big)}_{Q_\pi(s,a)}. \end{aligned} \tag{3.6}$$

Replacing the on-policy Value and Q-value functions in (3.6) with their optimal counterparts yields the *Bellman Optimality Equations*:

Whether the underlying model of the MDP is known or unknown, the *Bellman Optimality Equations* or their approximations can be used as update rules for estimating the Value or Q-value functions in iterative methods.

3.2.1 The MDP is known: Value Iteration

Assuming that the model of the MDP is known (i.e. known transition probabilities and a known reward function), *Value Iteration* (Sutton and Barto, 1998) can be used

$$
\begin{aligned}
V_*(s) &= \max_{a \in \mathcal{A}} Q_*(s,a), \\
Q_*(s,a) &= \sum_{s' \in \mathcal{S}} p(s'|s,a) \left(R(s,a,s') + \gamma V_*(s') \right), \\
V_*(s) &= \max_{a \in \mathcal{A}} \underbrace{\left[\sum_{s' \in \mathcal{S}} p(s'|s,a) \left(R(s,a,s') + \gamma V_*(s') \right) \right]}_{Q_*(s,a)}, \\
Q_*(s,a) &= \sum_{s' \in \mathcal{S}} p(s'|s,a) (R(s,a,s') + \gamma \underbrace{\max_{a' \in \mathcal{A}} \left(Q_*(s',a') \right)}_{V_*(s')})
\end{aligned}
\tag{3.7}
$$

Bellman Optimality Equations

to approximate the Value function. *Value Iteration* is an iterative method that uses the Bellman Optimality Equations (3.7) as an update rule for the value of the states. Given the current value estimate V_n for the potential following states, the new value estimate V_{n+1} of state s is given by:

$$
V_{n+1}(s) = \max_{a \in \mathcal{A}} \underbrace{\left[\sum_{s' \in \mathcal{S}} p(s'|s,a) \left(R(s,a,s') + \gamma V_n(s') \right) \right]}_{Q_n(s,a)}. \tag{3.8}
$$

Value Iteration loops over all states of the MDP to update their values until a convergence criterion is met. Once the Value function is learned, one can plan over the state space by selecting the action that maximizes the Q-value to recover the optimal policy.

Model-based RL approaches (Polydoros and Nalpantidis, 2017) (e.g. the Dreamer algorithm introduced in the header of this Section) assume that the MDP is unknown and try to learn an explicit model of it to plan. In this thesis, we only focus on Model-free RL methods which aim to learn a policy in an unknown MDP without learning any explicit model.

3.2.2 The MDP is unknown: from Temporal Differences to Deep Q-learning

In the context of Robot Learning, it is preferable to assume that the model is unknown. In this case, the Value function must be learned through sequences of interactions with the environment. These sequences of interactions are called trajectories or rollouts.

A seminal work in model-free RL is *Temporal Differences* (TD) learning. TD learning can be used to approximate the Value function from incomplete trajectories in the environment. The key intuition behind TD learning is to bootstrap the Value function of the current state $V(s)$ in the trajectory from an estimate of the future return $R(s,a,s') + \gamma V(s')$ called the *TD target*. Vanilla TD learning uses the following update rule to approximate the Value function iteratively:

$$V(s) \leftarrow V(s) + \alpha \left[\underbrace{R(s,a,s') + \gamma V(s')}_{\text{TD target}} - V(s) \right], \tag{3.9}$$

with learning rate hyper-parameter α. Here, the optimal Value function V_*(3.7) is approximated by the TD target.

A similar TD learning relation can be derived for the Q-value. Here, the TD-target approximates the optimal Q-value Q_* (3.7):

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[\underbrace{R(s,a,s') + \gamma Q(s',a')}_{\text{TD target}} - Q(s,a) \right]. \tag{3.10}$$

The Q-learning algorithm (Watkins and Dayan, 1992) can be derived from this last update rule. In Q-learning, the Q-value of the current state-action pair (s,a) is updated using a greedy version of (3.10) where the next action a' is selected according to the current estimate of the best action:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q(s',a') - Q(s,a) \right]. \tag{3.11}$$

While TD-learning and Q-learning are restricted to discrete state space, one can approximate the value of continuous states using function approximators. For instance, in Deep Q-Network (Mnih et al., 2013), a neural network is used to approximate the Q-value in continuous state spaces corresponding to 2D images of Atari games. However, exploring large and complex state spaces is a difficult challenge that requires a good balance between exploration and exploitation (Amin et al., 2021).

3.2.3 The exploration/exploitation dilemma

When the agent faces an unknown environment, a greedy *exploitation* of a randomly initialized Q-value is unlikely to result in a satisfying behavior. The agent must *explore* by testing different state-action combinations to collect additional information about

the environment, iterate over the Q-value and finally approach the optimal Q^*. However, the agent should not explore endlessly. It should balance the action selection between exploration and exploitation. This is referred to in the RL literature as the *exploration/exploitation dilemma*.

In easy-to-explore problems, the simple ϵ-greedy exploration strategy used in (Watkins and Dayan, 1992; Mnih et al., 2013) is enough to fully explore the environment. More sophisticated action selection mechanisms like entropy maximization or regularization (Haarnoja, Zhou, Abbeel, et al., 2018; Haarnoja, Zhou, Hartikainen, et al., 2018; Eysenbach and Levine, 2021) are necessary to solve more complex RL problems like robotics environments (Todorov et al., 2012) with high-dimensional continuous state and action spaces and complex dynamics.

3.2.4 Soft Actor-Critic

With the exploration/exploitation dilemma in mind, we shall derive the Soft Actor-Critic (SAC) algorithm, a state-of-the-art Deep RL algorithm, from the Q-learning algorithm described in Section 3.2.2. Remember that the Q-value update rule of Q-learning reminded below requires a "max" operation over the actions:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s,a,s') + \gamma \underbrace{\max_{a' \in \mathcal{A}} Q(s',a')}_{\text{tractable } max \text{ required}} - Q(s,a) \right]. \tag{3.12}$$

This operation is tractable if the action space is discrete (and finite). However, the operation must be approximated if the action space is continuous. In SAC, a parameterized policy $\pi_\theta(s)$ is used to approximate the operation $\arg\max_{a \in \mathcal{A}} Q(s,a)$ and select action a'.

Using parameterized approximations of the Value and Q-value functions to cope with continuous states and a parameterized policy to cope with continuous actions, we obtain an Actor-Critic algorithm where:

- The *Critic* corresponds to the approximated Value or Q-value functions. In SAC, the Critic includes a neural network Q_ω approximating the Q-value function and another network V_ψ approximating the Value function. Note that most recent implementations get rid of the latter network and use Q_ω to approximate the value function.
- The *Actor* corresponds to the parameterized policy $\pi_\theta(a|s)$ that approximates

the argmax operation over the Q-value function. In SAC, the actor is stochastic and corresponds to a neural network returning the mean and standard deviation of an isotropic Gaussian distribution, often augmented with a $tanh$ squashing function.

If the Critic and the Actor are differentiable parameterized functions (e.g. neural networks as in SAC), they can be trained using gradient-based optimization. Thus, a gradient step replaces the iterative update in (3.11).

The gradient can be computed using the latest collected transitions as in (Schulman et al., 2015; Schulman et al., 2017), which results in an *on-policy* algorithm. Otherwise, a *Replay Buffer* (noted $\mathcal{D}$ for *dataset*) can be used to memorize all the past transitions collected by the agent. A batch of those memorized transitions can then be sampled uniformly to perform the update in an *off-policy* fashion. This mechanism called *Experience Replay* was introduced in Deep Q-Network (Mnih et al., 2013) and is used in state-of-the-art Deep RL algorithms like DDPG (Lillicrap et al., 2015), TD3 (Fujimoto et al., 2018), SAC (Haarnoja, Zhou, Abbeel, et al., 2018) and TQC (Kuznetsov et al., 2020).

At this point, the underlying exploration mechanism is the only missing part to derive SAC. To encourage exploration in complex environments with continuous state and action spaces, SAC augments the usual RL objective with an entropy measure of the stochastic policy. This entropy measure of the stochastic policy is weighted by a parameter α_{ent} and added to the Q-value to construct the *Soft Q-value*.

$$
\begin{aligned}
Q(s,a) &= \mathop{\mathbb{E}}_{S'\sim p}\left[R(S,A,S') + \gamma V_\pi(S') | S=s, A=a)\right] \\
&= \mathop{\mathbb{E}}_{S'\sim p}[R(S,A,S') + \gamma \mathop{\mathbb{E}}_{A'\sim\pi}[Q(S',A') \underbrace{-\alpha_{ent}\log\pi(A'|S')}_{\text{weighted entropy measure}}] \\
&\quad |S=s, A=a] \\
&= \mathop{\mathbb{E}}_{\substack{S'\sim p \\ A'\sim\pi}}[R(S,A,S') + \gamma Q_\pi(S',A') - \alpha_{ent}\log\pi(A'|S') \\
&\quad |S=s, A=a)],
\end{aligned}
\tag{3.13}
$$

Soft Q-value

Using this augmented definition of the Q-value, regression objectives and their associated (stochastic) gradient can be derived for each of the elements of the Actor-Critic architecture. Note that SAC being an off-policy algorithm, the state and action random

variables are sampled from the Replay Buffer $\mathcal{D}$, which approximates the state and state-action distributions.

- V_ψ is trained to minimize the Mean Squared Error (MSE) between its current estimate of the Value function and the expectation over actions of the soft Q-value:

$$J_V(\psi) = \mathop{\mathbb{E}}_{S\sim\mathcal{D}}\left[\frac{1}{2}\left(V_\psi(S) - \mathbb{E}_{A\sim\pi}\left[Q_\theta(S,A) - \alpha_{ent}\log\pi_\theta(A|S)\right]\right)^2\right], \tag{3.14}$$

  using the unbiased estimator of the associated gradient:

$$\hat{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi(S) - Q_\theta(S,A) + \log\pi_\psi(A|S). \tag{3.15}$$

- Q_ω is trained to minimize the MSE between the TD target and the current estimate of the Q-value, which is called the *soft Bellman residual*:

$$J_Q(\omega) = \mathop{\mathbb{E}}_{S,A\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\omega(S,A) - \mathbb{E}_{S'\sim\rho_\pi(s)}\left[R(S,A,S') + \gamma V_\psi(S')\right]\right)^2\right], \tag{3.16}$$

  using an unbiased estimator of the associated gradient:

$$\hat{\nabla}_\omega J_Q(\omega) = \nabla Q_\omega(S,A)\left(Q_\omega(S,A) - R(S,A,S') - \gamma V_\psi(S')\right). \tag{3.17}$$

- Finally, the policy is trained to minimize the KL-divergence between the current policy and a distribution derived from the Q function.

$$J_\pi(\theta) = D_{KL}\left(\pi_\theta(.|S)\,||\,\frac{\exp Q_\omega(S,.)}{Z_\omega(S)}\right), \tag{3.18}$$

  with Z_ω an intractable normalization function that does not contribute to the stochastic gradient:

$$\hat{\nabla}_\theta J_\pi = \nabla_\theta\log\pi_\theta(A|S) + \left(\nabla_A\log\pi_\theta(A|S) - \nabla_A Q(S,A)\right)\nabla_\theta f_\theta(\epsilon;S), \tag{3.19}$$

  where $f_\theta(\epsilon;S)$ is the result of the reparameterization trick applied to the policy in order to make it differentiable (Kingma and Welling, 2013).

Therefore, an update rule of SAC corresponds to taking the gradients of these objectives and performing one step of gradient descent for each.

One should note that the implementation of SAC that we use in this thesis has a fixed weighting parameter for the entropy α_{ent}. However, a more recent variant of SAC includes an automatic temperature tuning mechanism not covered here (Haarnoja, Zhou, Abbeel, et al., 2018).

The combination of the exploration mechanisms of SAC and its smartly engineered implementation details (see (Haarnoja, Zhou, Abbeel, et al., 2018; Haarnoja, Zhou, Hartikainen, et al., 2018)) makes it one of the most efficient out-of-the-box approaches to solve complex RL problems involving continuous state and action space (Haarnoja, Zhou, Abbeel, et al., 2018). However, the update rules (3.14), (3.16)(3.18) require informative reward signals to guide the learning process. Thus SAC may still fail in the context of an ill-defined reward function including sparse or deceptive reward signals.

3.3 A well-defined reward function is crucial in RL

Even when equipped with action exploration mechanisms like an entropy bonus (3.13), RL algorithms still require a well-defined reward function to solve the RL problem. Indeed, even in a simple 2D maze navigation task, trajectories obtained by uniformly sampling actions are unlikely to explore the maze entirely (Matheron, 2020). In this context, the reward function should help the agent gradually explore the environment and, eventually, find the desired states. In particular, the reward function must be sufficiently *dense* so that an agent can find paths that yield a larger cumulative reward only using action exploration mechanisms (e.g. maximum entropy actions).

However, such a reward function can be difficult to design even for simple tasks. For instance, in the 2D maze presented in Figure 3.2, SAC fails to navigate toward the middle of the maze using naive *dense* or *sparse* reward functions. Indeed, the dense reward corresponds to the Euclidean distance to the middle of the maze and guides the agent towards local optima along the walls. Moreover, the reward being null in the corridors, it provides no learning signal to the agent.

Similarly, a reward function that is zero except in the desired states - a *sparse reward* function - may be equally problematic (Matheron, 2020). Indeed, in this context, the Value of each discovered state is likely to stay close to zero until the agent encounters the rewarded states by chance. Until then, the agent benefits from no guiding signal. Different options, requiring different levels of expert inputs, may be considered when facing such ill-defined reward functions. These options range from the hand-designed *Reward Shaping* to fully autonomous intrinsincally motivated agents.

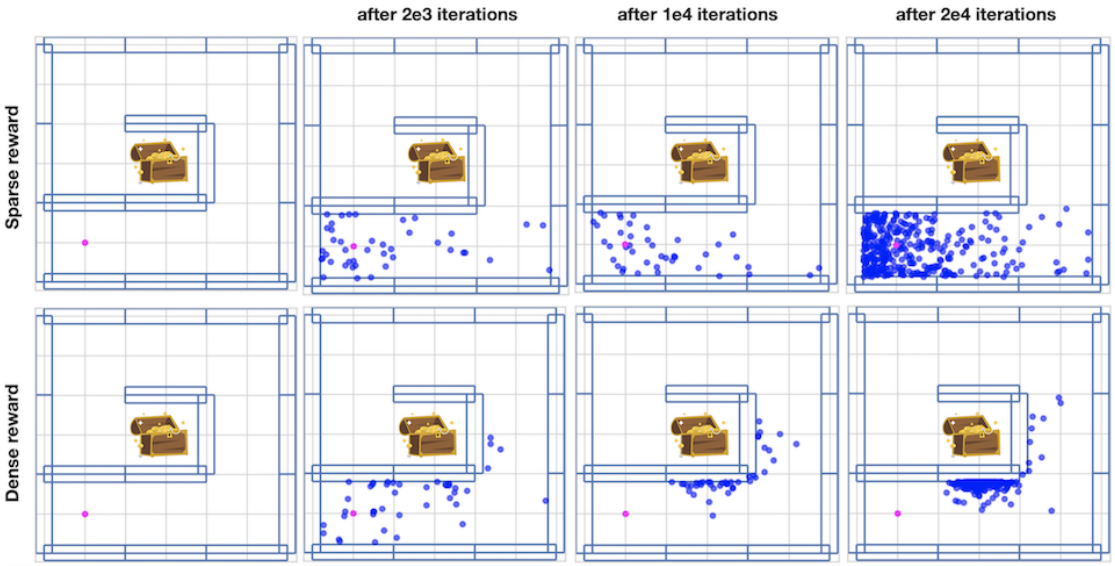


Figure 3.2: Training SAC with sparse and deceptive reward functions in a simple 2D maze where a particle-like agent moves using $(\Delta x, \Delta y)$ actions. Two naive reward functions are tested. **Top row:** the reward is zero everywhere except in the state where the treasure is. **Bottom row:** the reward corresponds to the Euclidean distance between the current state and the treasure. Blue dots represent the final state of the training trajectories started from the pink dot. When the reward is sparse, the agent is not guided by any learning signal and relies on the entropy bonus of SAC to explore. When the reward is dense, the agent gets stuck in local optima against the walls. In both cases, the agent fails to navigate the maze.

3.3.1 Reward Shaping

Reward shaping consists in augmenting the reward function with additional features guiding the agent toward the desired states. For instance, in the navigation task in Figure 3.2, a reward taking into account the walls and based on the shortest path between the entrance and the middle would allow the agent to avoid local optima. However, such reward shaping often requires non-negligible prior knowledge about the task which is not available in fully autonomous systems. Without such prior knowledge, an RL agent may rely on intrinsic motivations to explore the state space.

3.3.2 Intrinsic Motivations

According to psychologists and cognitive scientists from the field of Developmental Psychology (Berlyne, 1950; Berlyne, 1966; Gopnik et al., 2001; Oudeyer and L. Smith, 2016; Gottlieb and Oudeyer, 2018), infants and animals explore their environment for the sole purpose of experiencing something new and exciting. Inspired by this obser-

vation, several mechanisms have been developed by the RL community to encourage agents to perform *intrinsically motivated* (IM) actions i.e. actions performed to collect additional information about the environment rather than additional reward signals.

Novelty

A first approach to encourage the agent to take IM actions is *Novelty*. A novel state is a state that the agent has rarely or not visited at all. Thus, the novelty of the current state can be used to augment the reward function in order to encourage the agent to visit unseen states. To assess the novelty of a state, it is necessary to keep track of the previously visited states. In finite discrete state spaces, one can count the number of times each state has been visited (Brafman and Tennenholtz, 2002; Kearns and Singh, 2002). The lower the number of visits, the higher the novelty reward. However, such a counting mechanism is not compatible with continuous state spaces. A first alternative is to discretize the continuous state space with hash functions (Tang et al., 2017). Another is to use a density model to estimate the visitation distribution (Bellemare et al., 2016; Burda, Edwards, Storkey, et al., 2018). Novelty-based exploration has been successfully applied to several Hard-Exploration RL problems, including the challenging Atari game *Montezuma's revenge* (Bellemare et al., 2016). Nevertheless, one should note that a reward function augmented with a novelty bonus results in a non-stationary problem which often requires accurate hyper-parameter tuning to achieve the expected performance.

Prediction-based Intrinsic Motivations

An alternative approach to promote exploration is *Prediction-based Exploration*, also called *Curiosity*. Here, the agent is rewarded for improving his knowledge of the environment. A variant of this approach is inspired by the curiosity measure (Schmidhuber, 1991). The agent is equipped with a forward dynamics model to assess its knowledge about the environment (Stadie et al., 2015; Burda, Edwards, Pathak, et al., 2018). Using this $f : (s, a) \rightarrow s$ model, the agent estimates the future state that should result from the application of a given action in a given state. If the predicted next state $f(s, a)$ is different from the actual one s', the agent gains knowledge about the environment. Therefore, the error $|f(s, a) - s'|_2$ can be used to augment the reward similarly to novelty. Several variants of this prediction-based exploration mechanism have been proposed. In (Pathak et al., 2017), the error is computed with respect to an inverse forward model. In (Houthooft et al., 2016), tools from Information Theory (Shannon, 1948) are used to estimate the gain of knowledge. Indeed, the reward is augmented by entropy reduction in the probabilistic forward dynamics model.

Empowerment

Empowerment is another class of IM approaches based on Information Theory. Two seminal works in empowerment-based RL are Variational Intrinsic Control (VIC) (Gregor et al., 2016), and Diversity Is All You Need (DIYAN) (Eysenbach et al., 2018). Several extensions like Explore, Direct, and Learn (EDL) and UPSIDE have been proposed since (Hansen et al., 2019; Campos et al., 2020; Kamienny et al., 2021).

Each approach follows the same underlying strategy of encouraging exploration while forcing *controllability*. On the one side, the agent maximizes the entropy of the distribution over trajectories, which encourages exploration. On the other side, it minimizes the entropy of the distribution over trajectories when conditioned on distinctive inputs which can be interpreted as skills or options. Because the entropy of the distribution over trajectory is intractable without a known transition probability function, most empowerment-based methods rely on variational approximations. This limits their applicability to simple environments (Aubret et al., 2019).

3.4 Goal-Conditioned Reinforcement Learning

In Goal-Conditioned Reinforcement Learning (GCRL) problems (Kaelbling, 1993; Moore et al., 1999; Schaul et al., 2015; Nasiriany et al., 2019; Chane-Sane et al., 2021), the agent no longer tries to solve a single problem. Instead, the agent is trained to solve multiple problems defined by their associated goals. For instance, instead of training to reach a single goal corresponding to a unique position in a 2D maze navigation task, a goal-conditioned agent can be trained to reach many different goals corresponding to every possible position.

A GCRL problem extends the MDP framework with a goal space. The reward function is replaced by a goal-conditioned reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{G} \to \mathcal{R}$ which depends on the considered goal. One can define a dense goal-conditioned reward e.g. a reward function that is proportional to the distance to the goal. However, as explained in Section 3.3, a dense reward is likely to lead the agent towards local optima. Therefore, in most GCRL problems, the reward function is sparse and the agent only receives a reward for achieving the desired goal or for getting close enough to it according to a threshold ϵ.

$$R(s, a, s', g) = \begin{cases} 1 \text{ if } |s' - g|_2 < \epsilon \\ 0 \text{ otherwise.} \end{cases} \tag{3.20}$$

The discount factor is also modified in GCRL. Using the formulation of (Schaul et al., 2015), it is replaced by a goal-conditioned discount function $\gamma : \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{R}$ which sets the discount factor to 0 when the agent reaches the goal (Schaul et al., 2015):

$$\gamma(s, g) = \begin{cases} 0 \text{ if } s = g, \\ \gamma \text{ otherwise.} \end{cases} \tag{3.21}$$

Therefore, the RL objective can be rewritten for GCRL as follows:

$$\mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{\infty} R(S_t, A_t, S_{t+1}, g) \prod_{k=0}^{t} \gamma(S_k, g)\right]. \tag{3.22}$$

All the rewards received after reaching a goal are canceled. This makes transitions to goal states equivalent to transitions to terminal states, i.e. states terminating the agent trajectory.

3.4.1 Distance-based sparse reward

In a complex environment with a high-dimensional state space, reaching a goal corresponding to a precise state may be prohibitively difficult, thus it is common to consider goals that represent low-dimensional projections of states (Nachum et al., 2018; Bagaria, Senthil, Slivinski, et al., 2021; Ecoffet et al., 2021). To achieve a goal g, the agent must reach any element of the success state set $\mathcal{S}_g$, that is any state $s \in \mathcal{S}$ that can be mapped to a goal $g_s \in \mathcal{G}$ within a distance less than $\epsilon_{success}$ from g, $\epsilon_{success}$ being an environment-dependent hyper-parameter.

In practice, a state is mapped to a goal according to a projection $p_{\mathcal{G}} : \mathcal{S} \rightarrow \mathcal{G}$ associated with the definition of the goal space. In general, the common L2-norm is used to compute the distance between two goals. Besides, the environment-agnostic goal-conditioned reward function is generally defined as

$$R(s, a, s', g) = \begin{cases} 1 \text{ if } s' \in \mathcal{S}_g \\ 0 \text{ otherwise.} \end{cases} \tag{3.23}$$

3.4.2 Relabelling

As explained in Section 3.3, learning in a sparse reward context can be challenging for RL algorithms (Bellemare et al., 2016; Houthooft et al., 2016; Burda, Edwards, Storkey, et al., 2018). GCRL agents often use the Hindsight Experience Replay (HER) relabelling technique (Andrychowicz et al., 2017) to collect sparse rewards more often. During a trajectory, if the agent fails to reach the goal it is conditioned on, HER may relabel some transitions by replacing the initially intended goal with a goal it happened to achieve. This way, the agent can learn from failed trajectories by receiving rewards for accidentally achieving originally unintended goals.

3.4.3 Exploration in GCRL

While relabelling helps goal-conditioned agents learn how to reach local goals they accidentally achieved, it does not encourage them to explore.

A class of GCRL algorithms called *autotelic agents* have been specifically designed to help a goal-conditioned agent explore. An autotelic Agent in RL corresponds to a GCRL agent that generates its own goals (Colas et al., 2022). The goal selection procedure constitutes an automatic curriculum of goals. Easy-to-achieve goals should be selected first followed by increasingly difficult ones.

However, identifying easy-to-achieve goals is not an easy task. In GOAL-GAN (Florensa et al., 2018), a feasibility score is calculated for a given goal. This feasibility score corresponds to the empirical probability of success of the agent reaching the goal. A Generative Adversarial Network (GAN) (Goodfellow et al., 2014) is then trained to generate goals with an intermediate feasibility score.

Other approaches use IM for goal selection. Novelty-based goal selection biases selection toward goals considered novel. In several works (Pong et al., 2019; Pitis et al., 2020), a density model is trained on experienced goals and approximates the distribution of reachable goals. Using this model, one can sample goals outside the reachable set of goals, on its edge, or of intermediate difficulty Castanet et al., 2022. Another robust approach is the First Return, Then Explore algorithm (Ecoffet et al., 2021), which approximates a density model using a discrete exploration grid filled with the goals the agent achieved. Before starting a new training trajectory, the agent selects the goal corresponding to the least visited cell. It then tries to reach it and, eventually, explores from the advanced state resulting from the goal achievement. This approach achieved state-of-the-art scores in the challenging Montezuma's Revenge Atari game.

3.5 Can we use sampling-based MP algorithms to solve RL problems?

Remembering what a sampling-based MP algorithm is trying to solve and how it does it (see Chapter 2), one may wonder if these algorithms can be applied to solve an RL problem (Matheron, 2020) and, in particular, a Hard Exploration Problem. For instance, can one use RRT to explore the state space of an MDP and, as a consequence, solve an RL problem without relying on any reward function?

Several limitations arise when using a sampling-based MP algorithm out of the box to solve an RL problem. Three of them are discussed in the following paragraphs. The first one comes from the potentially high-dimensional search space. The second one results from the difficulty of expanding an exploration tree in a high-dimensional non-holonomic environment. The final one comes from the potential stochastic nature of the system.

3.5.1 High-dimensional search spaces

In an RL problem, the exploration tree must be constructed directly in the state space. As explained in Chapter 2, the state of a robot can be represented by a vector $s = (q, \dot{q})$ where q is the configuration of the robot. Therefore, the bigger the configuration space, the larger the dimension of the search space. For instance, in the Humanoid Mujoco physical simulator (Todorov et al., 2012), the state space has ~ 80 dimensions and observations are even bigger with 376 dimensions.

Moreover, without access to any controller, a node in the exploration tree can only be expanded using random actions which may also correspond to high-dimensional vectors. As a result, the applicability of sampling-based MP algorithms to solve an RL problem including high-dimensional states and actions may be limited by the curse of dimensionality (Bellman, 1966).

To cope with this limitation, one can reduce the search space size. For instance, in (Dalibard and Laumond, 2011), the authors use the linear dimensionality reduction property of Principal Component Analysis (Pearson, 1901) to identify expansion directions in the free configuration space. A second example is the Go-Explore algorithm (Ecoffet et al., 2019a; Ecoffet et al., 2021) which solves a manipulation task where the state space is embedded in $\mathcal{R}^{268}$ and the action space in $\mathcal{R}^8$ using an exploration phase which may be seen as a variant of the Ex algorithm (Matheron, 2020; Matheron et al., 2020). To do so, the authors reduced the search space using a hand-designed

mapping from the state space to a 4-dimensional space. Three dimensions correspond to the Euclidean position of the gripper, and the last indicates whether the object is grasped. Although their approach found a valid grasping trajectory, it required $\sim 2.5\ 10^6$ expansions, as most of them were *constrained expansions*.

3.5.2 Constrained expansions

As explained in Section 2.2.3, in a non-holonomic environment, finding a path between two close configurations may be a difficult problem as long trajectories may be required to connect them (see Figure 2.3), or, worse, it may be impossible to connect.

Let's imagine that one uses RRT to construct an exploration tree and solve an RL problem in a non-holonomic environment. As the state space is too large, a mapping to a smaller search space of interest is used as in Go-Explore to reduce the size of the search space (Ecoffet et al., 2021). In that case, it is possible that some of the nodes correspond to configurations that cannot be connected to any further configurations. Indeed, in some cases, a configuration can only be connected to already explored configurations. In other worse cases, a configuration may only be connected to itself. Then, expanding the tree from these nodes results in no further exploration. In this context, sampling-based MP algorithms may enter failing modes.

In fact, such failing modes may arise as soon as the search space corresponds to a subspace of the configuration space, even if the configuration space is a low-dimensional space. For instance, let's take a variant of the Reeds-Sheep car used in Figure 2.3 that cannot move backward. This system is called the Dubins car (Dubins, 1957). Growing an exploration tree in the $X \times Y$ space of 2D positions instead of the $X \times Y \times \Theta$ space which includes the orientation of the car may result in advanced (x, y) nodes that cannot be expanded further because the car is oriented toward a wall (see Figure 3.3).

In this example, one can simply consider the whole configuration space $X \times Y \times \Theta$ as the search space rather than $X \times Y$ to avoid these failure modes. However, a similar solution cannot be considered in an environment with a high-dimensional configuration space that requires a low-dimensional search space to resist the curse of dimensionality.

This problem of being unable to expand from certain robot configurations is a central element in the contribution part of this thesis.

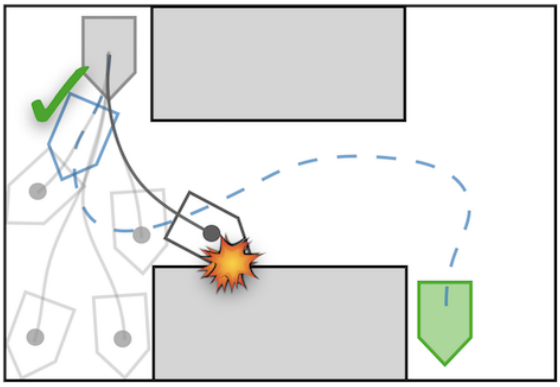


Figure 3.3: Example of a failure mode. An MP algorithm builds an exploration tree (in grey) in the $X \times Y$ space while the configuration space corresponds to $X \times Y \times \Theta$. The path to the desired configuration (in blue) requires a specific sequence of orientations to navigate the corridor (dashed blue line). However, the most advanced node in the corridor (dark gray) may correspond to a blocked configuration. Expanding from this node results in no further exploration.

3.5.3 Reset assumption

To expand a node in the exploration tree, the robot must return to the corresponding state in order to apply a random action. To reset the robot to that state, two approaches can be considered.

In the first option, one assumes that the robot can be reinitialized in the states it has already visited. Although this assumption is unrealistic in the context of a physical robot, it has already been exploited in various forms in the RL literature (Salimans and R. Chen, 2018; Ecoffet et al., 2019a).

In the second option, one assumes that the robot can return to this state by replaying the sequence of actions used to extend the tree from this state. This would assume that the environment is fully deterministic. However, it is well-known that robotics environments are usually non-deterministic because of the noise of the sensors and the actuators.

A closed-loop control is then required to effectively return to the desired position. In a minimal context where no model of the system is available and control is more complicated than position or velocity control, it is impossible to use a low-level controller such as a Proportional Derivative Controller or a Linear Quadratic Regulator to do so. In this case, RL can be used to learn primitive skills between successive nodes (Faust et al., 2018; Bagaria, Senthil, and Konidaris, 2021) and eventually ensure such return.

3.6 Summary

Reinforcement Learning algorithms share a common foundation: the Markov Decision Process framework, the Values and Q-value functions and the Bellman equations. Using these key concepts and a few additional mechanisms (function approximations, experience replay,...), one can derive state-of-the-art algorithms like the Soft Actor-Critic algorithm. Despite being a robust algorithm, SAC cannot tackle Hard-Exploration Problems on its own. Therefore, several approaches taking inspiration from the theory of Intrinsic Motivation have been proposed to encourage the agent to explore the environment and tackle these problems.

The framework of RL has been extended with explicit goal conditioning to address multiple problems simultaneously. In Goal-conditioned RL (GCRL), the reward function depends on the targeted goal. As this reward function is often sparse, a vanilla GCRL agent may struggle to explore the environment. For this reason, a GCRL agent often uses relabelling to densify the reward signals. In addition, autotelic GCRL agents adapt their goal selection procedure to explore.

Because of several limitations like the high dimensional search space, the constrained expansions and the required reset assumption, it is clear that using MP algorithms to solve an RL task is not a robust solution. In the second part of this thesis, we show that these limitations can quickly reappear in diversity search and RL methods, particularly those related to constrained expansions.

4 Background on Neuro-Evolution

Evolutionary Algorithms (EA) can be applied to a wide variety of optimization problems (Eiben and J. E. Smith, 2015) ranging from chemistry reactions (Gutierrez et al., 2014) to complex modeling (Ashlock, 2006). The all-around nature of EAs comes from the fact that they require minimal assumptions to be applied. In particular, they assume no access to the inner structure of the optimization problem. For instance, unlike gradient-based optimization methods, the assumption that the cost function to optimize is differentiable is not necessary: EAs are *black-box* optimization algorithms (Eiben, J. E. Smith, et al., 2003). Therefore, when cast into an MDP, EAs offer an interesting alternative to RL algorithms (Salimans et al., 2017; Such Petroski et al., 2018).

This chapter introduces some background on EAs, emphasizing Neuro-Evolution and Diversity Search. While Neuro-Evolution uses Evolutionary Algorithms to optimize the weights of parameterized policies (e.g., a neural network) (Stanley et al., 2019), Diversity Search uses Neuro-Evolution to obtain a set of policies resulting in diverse behaviors (Cully and Demiris, 2018b). The latter is particularly useful in the absence of well-defined learning signals, for example, if the reward is sparse or deceptive in an RL problem.

4.1 Evolutionary Algorithms to solve RL problems

Evolutionary Algorithms (EA) are a family of optimization algorithms inspired by the Theory of Evolution (Darwin, 1859). The core idea behind EA is to iteratively apply the concept of *survival-of-the-fittest* to generate a set of solutions called the *population*. At each iteration of an EA, the solutions contained in the population are evaluated on the problem to optimize. Based on their performance, the top-performing (or fittest) solutions have a higher chance of being selected while poor solutions are likely

to be eliminated. The selected solutions are modified or combined to form a new population eventually containing better solutions. The modification of a solution and the combination of two solutions are respectively called *mutation* and *cross-over* operations. These operations are inspired by the recombination and mutation of DNA sequences during reproduction (Mendel, 1865).

This optimization loop is illustrated in a 2D search space in Figure 4.1.

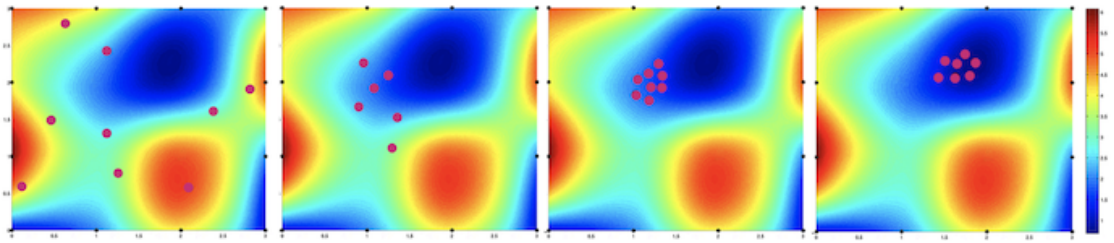


Figure 4.1: Running an Evolutionary Algorithm to find the minimum of a function in a 2D search space. At each iteration, the solutions are evaluated using the fitness function. The fittest solutions are more likely to be selected. Once selected, a solution is modified or combined with another one to yield a new, potentially better, solution. After ~ 20 iterations the population of solutions converges to the minimum of the function.

To evaluate the solutions in the population, an EA relies on a fitness function that indicates their performance at solving the problem. For instance, this fitness function may correspond to the minimization of a cost function as in Figure 4.1.

To avoid early local optima in the optimization process, the selection of the fittest elements of the population is often probabilistic. It can be performed in multiple different ways (Blickle and Thiele, 1996). In *Roulette-Wheel selection*, the probability of a solution being sampled in the population is proportional to its fitness. In *Tournament-based selection*, a subset of solutions is sampled uniformly from the population. The fittest solution wins the tournament and is selected. Overall, the intuition behind EA is that by selecting the fittest elements of the population, the population climbs (or slides over) the fitness landscape toward the optimum value.

When cast into an MDP to solve an RL problem, the population of solutions often corresponds to a set of parameterized policies (e.g. neural networks). During a policy evaluation, a trajectory is collected in the state space of the MDP using the policy. The fitness of the policy is then computed according to the trajectory. For instance, a suitable fitness function may correspond to the sum of discounted rewards collected over the trajectory (Salimans et al., 2017; Such Petroski et al., 2018).

4.2 Evolving Neural-Networks

In Neuro-evolution, an EA is used to evolve a population of policies corresponding to neural networks (Floreano and Mondada, 1994) (Whitley et al., 1990). While a more complex encoding can be considered (Stanley et al., 2009; Templier et al., 2021), a neural network is often encoded as a vector resulting from concatenating its weights (Such Petroski et al., 2018). During a mutation operation, the values of the vector encoding the policy are randomly perturbed. Usually, such perturbations are done using Gaussian noise or Polynomial mutations (K. Deb et al., 2002).

When a neural network is encoded as a vector of weights, the search space size increases with the size of the network. For instance, a neural network with a usual RL architecture of two hidden layers containing 256 neurons each has more than $1e4$ parameters, which results in a search space with more than $1e4$ dimensions. Due to the curse of dimensionality (Bellman, 1966), exploring a large search space using random mutations can be extremely difficult. Different classes of methods have tackled this problem. In Policy Manifold Search (Rakicevic et al., 2021), the authors propose to learn a latent representation of the parameter space to reduce the dimension of the search space. A similar approach was proposed in Data-Driven Encoding MAP-Elites (Gaier et al., 2020). Another class of algorithms at the frontier of EA and RL exploits the differentiable nature of neural networks. It combines NE with gradient-based policy optimization to speed up the policy search (Pourchot and Sigaud, 2018; Nilsson and Cully, 2021; Pierrot et al., 2022; Sigaud, 2022). Finally, NE has been conducted using Evolution Strategies, a particular class of EAs. The resulting algorithms (Salimans et al., 2017; Colas et al., 2020) successfully solved complex robotics tasks, including humanoid locomotion tasks, with limited sample-inefficient.

The cross-over operation, which is fundamental to promoting diversity in the evolution of a population, is often discarded in Neuro-Evolution. Indeed, because of their non-linear nature, combining two neural networks by mixing their weights often results in a third neural network corresponding to a non-exploitable policy. An alternative to the naive combination of weights is based on the imitation of the outputs of both networks (Gangwani and J. Peng, 2017). However, the need for additional interactions with the environment to perform the cross-over operation limits its applicability when sample efficiency is required.

4.3 Diversity Search

Similarly to RL algorithms, EAs require that the reward function and, consequently, the fitness function are well-defined to resolve the underlying RL problem. For instance, if the reward function is sparse and trajectories resulting from the evaluation of randomly initialized policies collect a sum of rewards equal to zero, EAs cannot exploit any learning signal during the selection operation. EAs may also be penalized by deceptive reward functions. Similarly, returning to the example of the 2D maze navigation task given in Section 3.3, a fitness function based on a reward corresponding to the Euclidean distance to the exit is likely to result in a population of policies stuck in the same local optima presented in Figure 3.2.

In response to these obvious limitations of EAs when dealing with ill-defined fitness functions, Diversity Search (DS) approaches like Novelty Search (NS) (Lehman and Stanley, 2011a) and (Intrinsically Motivated) Goal Exploration Processes (GEP) (Benureau and Oudeyer, 2016; Forestier and Oudeyer, 2016; Forestier, 2019) have been developed by taking inspiration from the literature of Intrinsic Motivation (see Section 3). Instead of directly searching for a high-performing policy, these approaches search for policies with diverse behaviors, hoping that the desired behavior emerges at some point.

4.3.1 Novelty Search

In NS, the task-related fitness is abandoned and is replaced by a *novelty* metric. An Outcome Space $\mathcal{O}$ (also called Behavior Space) is defined to compute the novelty of the behavior of a policy. This space, which is usually hand-designed, summarizes the key features of a trajectory resulting from a policy evaluation. For instance, in a navigation task, an outcome may correspond to the final positions of the robot at the end of its trajectory (Lehman and Stanley, 2011a; Lehman and Stanley, 2011b).

An outcome is considered novel only if it differs from the previously observed behaviors. To assess the novelty of outcome $o_{new} \in \mathcal{O}$, NS uses an archive $\mathcal{A}_{novelty} = \{o_i\}_{i \in [0, N_{size}]}$ to memorize all the behaviors observed during the previous iterations. The novelty score then corresponds to the average distance to the K-nearest neighbors in $\mathcal{A}_{novelty}$ in the outcome space:

$$N(o_{new}) = \frac{1}{N} \sum_{i \in I_{NN}} \texttt{dist}(o_{new}, o_i), \tag{4.1}$$

where I_{NN} corresponds to the indices of the K-nearest neighbors. One should note that the outcome space is usually designed as a low-dimensional space. Therefore, the Euclidean distance is usually the selected distance (Aggarwal et al., 2001). Moreover, the archive can be organized as a k-d tree for efficient nearest-neighbor search (Bentley, 1975b).

In MAP-Elite (Mouret and Clune, 2015; Cully and Demiris, 2018b), the novelty evaluation is simplified. The archive is organized as a grid dividing the outcome space into cells. A surrogate novelty score then corresponds to the density of filled neighboring cells.

4.3.2 Goal-Exploration Processes

While sharing the same objective as NS, GEP proceeds differently to obtain policies exhibiting diverse behaviors. In GEP, the population and the archive are confounded. The population keeps track of all evaluated policies and their outcomes. A policy is selected as follows: an outcome is sampled in the outcome space, and the policy in the population with the closest outcome according to a distance in the outcome space is selected. While both NS and GEP assume that a low-dimensional outcome space equipped with a distance is defined, an additional assumption is made in GEP as it assumes that the outcome space can be sampled. Depending on the sampling strategy, the population may vary differently.

4.3.3 Is Neuro-Evolution compatible with maximum entropy?

As described in Section 3.3.2, RL methods also encourage exploration by maximizing the entropy of stochastic policies. Therefore, one could wonder if an analogous approach could be used for NE-based DS.

Remember that the behavior of a policy is computed according to the trajectory resulting from the policy evaluation. If the policy (or the environment) is stochastic, multiple evaluations should result in different trajectories and, therefore, different outcomes. Such non-deterministic outcomes could interfere with the selection operations. Indeed, a poor policy that gives a new outcome during a lucky trajectory will be selected with a high probability, although it is not a promising stepping stone. Thus, a common assumption in NE is that the policy and the environment are deterministic. As a result, NE cannot be combined with an action entropy maximization for DS.

4.4 Summary

EAs are based on the elementary concept of survival of the fittest. An EA optimizes a population of solutions by selecting and varying the best solutions.

EAs evolving a population of policies encoded by neural networks are called Neuro-Evolution (NE) algorithms. In NE algorithms, to evaluate the performance of a policy, the latter is used to collect a trajectory in the environment. The fitness is then computed according to the trajectory. For instance, the fitness function may correspond to the sum of rewards collected along the trajectory.

As in Reinforcement Learning, to tackle Hard Exploration Problems, NE algorithms search for diverse behaviors using mechanisms inspired by Intrinsic Motivation. Several Diversity Search algorithms have been proposed in the literature. Novelty Search and Goal Exploration Processes are among the most commonly used. Both approaches characterize the trajectory associated with a policy using low-dimensional outcomes. In Novelty Search, a novelty score is computed for each outcome and the most novel policies are more likely to be selected. In Goal Exploration Processes, the policy associated with the outcome closest to a uniformly sampled one is selected.

In Chapter 6, we discuss in detail the difference between these two DS algorithms. In particular, we show how their different selection mechanisms impact the dynamic of the population.

5 Learning from demonstrations

If the exploration mechanisms introduced in Section 3 and the Diversity Search methods presented in Section 4 are insufficient to tackle a Hard Exploration Problem, one can collect demonstrations of the desired task and learn from these demonstrations. Thus, by providing the agent with examples of how the task should be performed, it can leverage tools from Imitation Learning (IL) and learn more quickly and accurately than it would by trial and error. In IL, a policy is not trained to maximize any (potentially ill-defined) reward signal. Instead, the policy is trained to reproduce the behavior of an expert, typically by learning from example data. IL can be divided into three categories: non-interactive methods, Interactive Imitation Learning and RL-based methods (Celemin et al., 2022).

5.1 Non-Interactive Imitation Learning

IL is non-interactive when it only requires demonstrations and no additional signal or feedback during training. Non-interactive IL usually involves Supervised Learning. The goal is to produce a policy that can reproduce the same outputs as the expert for a given set of inputs. While the resulting agent does not necessarily understand the underlying process that generates the data, IL can still be useful when it is not possible or practical to train the agent in the environment, such as when the environment is dangerous or when no simulator is available.

The most popular example of non-interactive IL is Behavioral Cloning (BC) (Pomerleau, 1991). BC uses regression to imitate demonstrated actions in demonstrated states greedily. Although very simple, in specific contexts where the environment is not too noisy, a large amount of near-optimal expert data is available and the planning horizon is short, BC may be a satisfying approach (A. Kumar et al., 2021). However, the imitation policy often suffers from poor generalization outside the training dis-

tribution. Therefore, if compounding errors drive the agent into out-of-distribution states, the latter will likely select poor actions and fail to solve the demonstrated task. This problem is often referred to as a distribution shift problem (Ross et al., 2011).

Non-Interactive Imitation Learning also includes approaches based on Robot Programming from Demonstration (Billard et al., 2008). In these approaches, deterministic (Neumann and Steil, 2015; Perrin and Schlehuber-Caissier, 2016; S. Gupta et al., 2022) or probabilistic (Calinon et al., 2010; Silvério et al., 2018) models of the system's dynamics are constructed using demonstrations. A controller is then derived from this model. While initially restricted to low-dimensional state spaces, these approaches have been scaled to higher dimensions using dimension reduction techniques (Calinon and Billard, 2007). They are very interesting as they benefit from theoretical (global asymptotic or probabilistic) convergence guarantees (Ravichandar et al., 2020) lacking in Deep RL.

5.2 Interactive Imitation Learning

If available, one can rely on additional expert feedback during training to limit the impact of compounding errors. In Dataset Aggregation (Dagger) (Ross et al., 2011) when compounding errors make the agent shift from the training distribution, additional feedback is given by the expert to bring the agent back on the support of the training distribution.

Methods like Dagger that involve the teacher in the learning loop fall within the framework of Interactive Imitation Learning (Celemin et al., 2022). While their applicability is limited by the need for an expert to supervise the entire learning process, these approaches have been successfully applied to complex physical robotic tasks (Jauhri et al., 2020; Mészáros et al., 2022).

Nevertheless, one should note that it may be difficult for an expert to provide feedback on low-level actions (e.g. on precise torque control) (Ravichandar et al., 2020). Therefore, teachable agents based on GCRL and using a high-level goal representation such as binary predicates or natural language offer interesting possibilities for extending interactive Imitation Learning approaches (Colas, 2021; Akakzia et al., 2022).

5.3 RL-based Imitation Learning

Imitation Learning can leverage tools from RL to efficiently learn a policy from optimal and sub-optimal datasets of transitions.

RL-based Imitation Learning includes Offline RL (Prudencio et al., 2022), Inverse RL (Arora and Doshi, 2021), and RL from demonstrations (Schaal, 1996). These different classes of algorithms differ in their assumptions. For instance, Inverse RL algorithms assume no access to a reward function. On the other hand, an Offline RL agent does not interact with the environment.

5.3.1 Offline Reinforcement Learning

Similarly to BC, in Offline RL, an agent uses previously collected data to learn and improve its behavior without direct interactions with the environment and additional feedback from the expert. However, in offline RL, unlike IL, the data collected previously are not necessarily expert demonstrations. For instance, they can correspond to exploration trajectories. Offline RL aims to generate new behaviors rather than imitate the demonstrated behavior.

A seminal work in offline RL is Reward Weighted Regression (RWR) (Peters and Schaal, 2007). RWR is particularly close to BC. Indeed, the main difference is that RWR weights the policy regression for a state action pair using the associated (transformed) reward. Using this weighted term, RWR is able to learn from sub-optimal data.

Recent offline RL possesses additional interesting additional properties compared to BC (A. Kumar et al., 2021; Prudencio et al., 2022). Indeed, using tools like the Value function, Offline RL agents learn about the long-term consequences of their actions while BC and RWR learn to mimic an action at a given time only. An example of the effective use of the Value function is the Advantage Weighted Regression (X. B. Peng et al., 2019), an extension of RWR that weights policy regression according to the reward and the value. In addition, using poor data, Offline RL agents not only learn the optimal actions but also to avoid sub-optimal actions. Thus, relying on Bellman Equations, Offline RL can learn from optimal segments of sub-optimal trajectories rather than from complete optimal trajectories as in BC.

However, similarly to BC, Offline RL may suffer from a distribution shift in action selection during training. This distribution shift is caused by poor generalization outside the training dataset, which may result in a value overestimation of out-of-distribution state-action pairs (Prudencio et al., 2022). Most Offline RL approaches try to limit this distribution shift either using policy constraints (Fujimoto et al., 2019; A. Kumar et al., 2019; Fujimoto and Gu, 2021), critic regularization (Nachum et al., 2019), or uncertainty estimation (Agarwal et al., 2020).

Moreover, one should note that large training datasets are required to perform offline

reinforcement learning efficiently. Collecting such a large training dataset may be challenging in complex robotic contexts (Ravichandar et al., 2020).

5.3.2 Inverse Reinforcement Learning

Another way to leverage demonstrations using RL is Inverse Reinforcement Learning (IRL) (Arora and Doshi, 2021). An IRL algorithm is based on a two-step training loop. First, a reward function is learned using demonstrated transitions along with training transitions. Then, an agent is trained using this learned reward function and collects additional information about the environment. Using this additional knowledge, the two steps are repeated. In early IRL approaches, the reward was inferred by searching for a trajectory distribution that matches the same hand-designed features as the demonstrations (Ziebart et al., 2008). In more recent methods like Generative Adversarial Imitation Learning (GAIL) (Ho and Ermon, 2016b), the feature matching objective is replaced by an occupancy measure matching between the state-action distributions of the agent and the expert. While a large number of training episodes are required for IRL approaches to converge (Kober et al., 2013), this distribution-matching approach has been adapted to the off-policy setting in Discriminator Actor-Critic (DAC) (Kostrikov, Agrawal, Levine, et al., 2018), resulting in a significantly improved sample efficiency. Note that DAC efficiently solved complex simulated robotics tasks using a handful (~ 10) of demonstrations.

5.3.3 Reinforcement Learning from demonstration

RL from demonstration is a set of RL-based methods that leverage demonstrations. Unlike offline and inverse RL, they also interact with the environment and do not attempt to learn an explicit reward function. In RL from demonstrations, the dataset of expert interactions with the environment is continuously used to update the policy along with training interactions. They can be used to guide action selection during policy updates (A. Nair et al., 2018; Goecks et al., 2019), or to learn a more accurate value function faster (Vecerik et al., 2017; Paine et al., 2019; Reddy et al., 2019). Primal Wasserstein Imitation Learning (PWIL) (Dadashi et al., 2020) proposed to solve a distribution matching problem similar to GAIL and DAC without using Inverse RL. Instead, PWIL uses a hand-designed reward based on the Wasserstein distance between the state-action distributions of the expert and the agent. PWIL manages to imitate a complex locomotion behavior for an under-actuated humanoid in simulation using a single demonstration with unprecedented sample efficiency.

5.4 Summary

Imitation Learning can be divided into three categories: non-interactive methods, Interactive Imitation Learning and RL-based IL.

Non-Interactive Imitation Learning algorithms attempt to reproduce the same actions as those demonstrated by the expert without additional interaction with the environment. They assume that the demonstrations are optimal and may suffer from distributional mismatches.

In interactive approaches, an expert supervises the training of the agent and gives additional feedback in case of a distribution shift. While limited by the required availability of the expert, these approaches are able to learn complex behaviors with unmatched efficiency.

RL-based approaches can be divided into three subclasses. First comes Offline RL. Similarly to non-interactive methods, Offline RL does not assume access to the environment during training. In addition, by using RL tools such as reward and value functions, Offline RL algorithms can learn from sub-optimal data. However, similarly to Behavioral Cloning, Offline RL suffers from a distributional shift. Secondly, in Inverse RL, a reward function is learned by solving a distribution matching problem between the state-action distribution of the agent and the expert. The learned reward function is then used to train an RL agent. As these approaches often rely on generative architecture, they tend to suffer from poor sample efficiency and unstable training. Finally, in RL from demonstration, the dataset of expert demonstration is continuously used during the training procedure and guides the agent toward interesting states and optimal behaviors.

Part II Contributions

In brief

This second part of this manuscript presents our different contributions. Chapter 6 first analyses the difference in performance of two diversity search methods, Novelty Search and Goal Exploration Processes. The study reveals that these methods struggle in front of non-localities in the mapping between the parameters of the agent and its trajectories in the environment. Chapter 7 then proposes to decompose the corresponding hard exploration problems into simpler local exploration sub-problems. Although effective in simple holonomic environments, limitations arise when NSSC constructs a chain of partially characterized behaviors in an environment with more complex dynamics. In particular, this chapter introduces the notion of invalid exploration states, which are false stepping stones for exploration. With this limitation in mind, Chapter 8 addresses the hard exploration problems of interest by learning from a single demonstration. It proposes to decompose these problems into several simpler goal-reaching sub-problems. This chapter introduces the fundamental concept of valid success states, which highlights that when an agent tries to reach low-dimensional goals in a high-dimensional environment successively, it must be careful to reach each goal via states compatible with the achievement of the next goals. We propose two mechanisms to encourage the agent to reach these valid success states. They both lay the foundation of the DCIL-I algorithm proposed in Chapter 8, which addresses the problem of achieving hard-to-reach goals leveraging a single demonstration. In Chapter 9, the two mechanisms of DCIL-I are integrated into an original Goal-Conditioned Reinforcement Learning framework where states are extended to include the sequence of low-dimensional goals. Based on this framework, we propose an evolution of DCIL-I called DCIL-II. This more efficient variant can learn complex locomotion behaviors with unprecedented efficiency. Finally, Chapter 10 attempts to remove the main assumption of DCIL-I and DCIL-II, which assume that the agent can be reset in any state of the demonstration. The SR-DCIL variant, which only assumes a reset in one state, shows moderate results depending on the content of the demonstration. Nevertheless, it provides a solid basis for future work.

6 Analysis of the limitations of Diversity Search Neuro-Evolution

An artificial agent that relies on a sparse or deceptive reward signal to solve a Reinforcement Learning problem lacks trustful information to steer its learning process. This often results in complete task failure or poor performance (Matheron et al., 2019). Diversity Search has been successfully applied to learn control policies in Hard Exploration Problems (Lehman and Stanley, 2011a; Forestier and Oudeyer, 2016). To do so, Diversity Search algorithms rely on different forms of Intrinsic Motivation, such as novelty search or goal exploration (see Section 4). However, despite these successes, some fundamental difficulties limit their applicability. Indeed, these methods often require many interactions with the environment to achieve diverse behaviors. This budget becomes even more significant if we perform Neuro-Evolution to learn neural network policies.

In this chapter, we empirically evaluate the ability of Diversity Search algorithms to solve Hard Exploration Reinforcement Learning Problems. In particular, we highlight their poor sample efficiency due to the fragility of the mutation operator by drawing a parallel with sampling-based Motion Planning algorithms.

We can distinguish two classes of algorithms to combine Neuro-Evolution and Diversity Search: Goal Exploration Process (GEP) (Benureau and Oudeyer, 2016; Forestier and Oudeyer, 2016; Forestier, 2019) and Novelty Search (NS) (Lehman and Stanley, 2011a). Both classes have been applied to solve a variety of RL problems (Salimans et al., 2017; Such Petroski et al., 2018). The same observation was made on each occasion: although very powerful and easily parallelizable, these approaches are very sample-inefficient. Several methods combining them with RL algorithms have been proposed in an attempt to increase their efficiency (Colas et al., 2018; Cideron et al., 2020; Nilsson and Cully, 2021; Sigaud, 2022).

In this Chapter, we propose no miraculous recipe. Instead, we investigate the prop-

erties of these two classes of algorithms. In the first part, based on a very general *selection-expansion* framework, we reveal a similarity between these algorithms and Motion Planning (MP) algorithms like Expansive Spaces Trees (EST) (D. Hsu et al., 1997) and Rapidly-exploring Random Trees (RRT) (Lavalle, 1998). In the second part, we empirically compare both algorithms in two environments where a smoothness assumption on which MP algorithms implicitly rely either holds or not. We show that diversity algorithms are highly dependent on the design of the outcome space where the search for diversity is performed and that the smoothness of the mapping between the policy parameter space and the outcome space plays a crucial role in their search dynamics. In particular, we show that if the mapping is smooth enough, GEP and NS inherit the exploration properties of their MP counterparts and GEP outperforms NS. By contrast, if it is not, which is the usual case, NS and GEP perform differently and their performance strongly depends on specific heuristics, notably filtering mechanisms that discard some of the explored policies.

6.1 Selection-Expansion: a unifying framework for Diversity Search and sampling-based Motion-Planning algorithms

In this section, we highlight that NS and GEP share properties with two well-known MP algorithms, EST and RRT. To establish the similarity between both families of algorithms, we start from a more general framework that we call *selection-expansion* algorithms.

6.1.1 Selection-expansion algorithms

Imagine an agent searching in some space and looking for an area it knows nothing about. What should it do? The most classical approach is to keep a memory of what has already been explored, and to progress locally, i.e. by reconsidering previous trajectories or behaviors, and by expanding or slightly modifying them to find new areas of the space to explore. This is the basis of virtually all sampling-based motion planning algorithms, and the core mechanism of Go-Explore (Ecoffet et al., 2019a). We call this kind of algorithms selection-expansion algorithms because they share the common structure of maintaining an archive of previous samples and iterating over a sequence of two operators:

- the **selection operator** that chooses in the archive a sample from which to

expand;

- the **expansion operator** that adds one or several new samples built from the selected sample.

Usually, selection and expansion operators are designed to efficiently expand the frontier of explored areas towards unexplored regions of the space. To do so, there are two popular selection strategies. One can either:

Strategy 1 *rank all elements in the archive in terms of distance to their neighbors, and preferentially select those far away from their neighbors, which suggests that they lie in a region with a low density of exploration; or*

Strategy 2 *randomly draw a sample anywhere in the search space and select the closest sample in the archive. This way, samples which are close to large unexplored regions have a higher chance of being selected.*

In the next section, we describe applications of the above selection-expansion algorithms in two domains, namely Motion Planning and Diversity Search algorithms. This reveals a striking similarity between both families of algorithms.

6.1.2 Application to Motion Planning

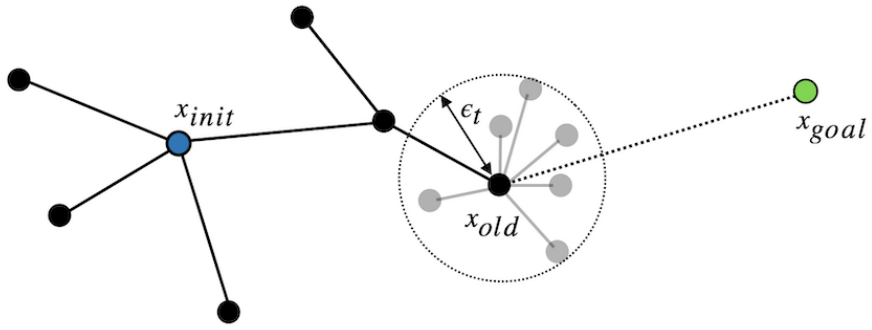


Figure 6.1: Expansion operators in motion planning with unknown dynamical systems. When the dynamical system is unknown, random controls are propagated through the system, yielding random nodes contained in a local ball.

Sampling-based MP algorithms use selection-expansion algorithms to build an exploring tree eventually containing a path from the starting configuration to the goal.

In the MP context, the need for a local expansion operator comes from the fact that the system must navigate locally from its current configuration to the next using a few control steps only.

Thus, the expansion operators of the “model-free" MP algorithms considered here typically perform random actions from the selected configurations to reach a new configuration which is then added to the exploration tree.

For the selection operator, there exist MP algorithms corresponding to both strategies described above.

Expansive Spaces Trees

The Expansive Spaces Trees (EST) algorithm corresponds to a family of algorithms where the selection operator uses Strategy 1. These algorithms select the most isolated nodes based on an estimate of the local density of nodes. Various approximations of the local density can be used. For instance, a node can be selected based on its number of neighbors within a certain range D. The nodes are selected with a probability distribution based on the weight of nodes so that the nodes with fewer neighbors tend to be selected with higher frequency than others.

Other estimates of the local density of nodes can also be used. In this chapter, we consider the mean distance to the K-nearest nodes as an estimation of the local density. Given a set of N nodes $S = \{s_i\}_{[1,N]} \in C_{free}^N$ and a set of K nearest-neighbors $\{\mu_1, ..., \mu_k\} \subset S$ associated to node s_n, the latter has a weight :

$$w_n = \frac{1}{k} \sum_{i=1}^{k} dist(s_n, \mu_i). \tag{6.1}$$

The probability p_n for s_n to be selected is proportional to its weight:

$$p_n = \frac{w_n}{\sum_{i=0}^{N} w_i}. \tag{6.2}$$

Besides, in the general case without specific knowledge on the system, a random control input is used during one or a few steps to expand s_n to a new state s_{new}. If no collision occurs, s_{new} is added to the tree.

Rapidly-exploring Random Trees

Like EST, Rapidly-exploring Random Trees (RRT) is a sampling-based path-planning algorithm. But, in contrast to EST, RRT performs selection according to Strategy 2. It draws a random goal configuration s_{samp} and selects the closest node in the set of already visited nodes. Note that sampling a random configuration requires determining the boundaries of the space where to sample from, a stronger prerequisite than in EST.

Given a set of nodes $\{s_i\}_{i\in[1,N]} \in C_{free}^N$, one can define the *Voronoi diagram* of these points as a set of *Voronoi cells* with one Voronoi cell per point, where the Voronoi cell of each point s_i is the subspace of all points that are closer to s_i than to any other point of the set. When selecting randomly, the probability p_k for an already visited node s_k to be selected is proportional to the volume of its Voronoi cell:

$$p_k = \frac{volume(Voronoi\ cell\ s_k)}{\sum_{i=0}^{N} volume(Voronoi\ cell\ s_i)}. \tag{6.3}$$

After selection, without knowledge of the system, expansion is also performed by applying a random control.

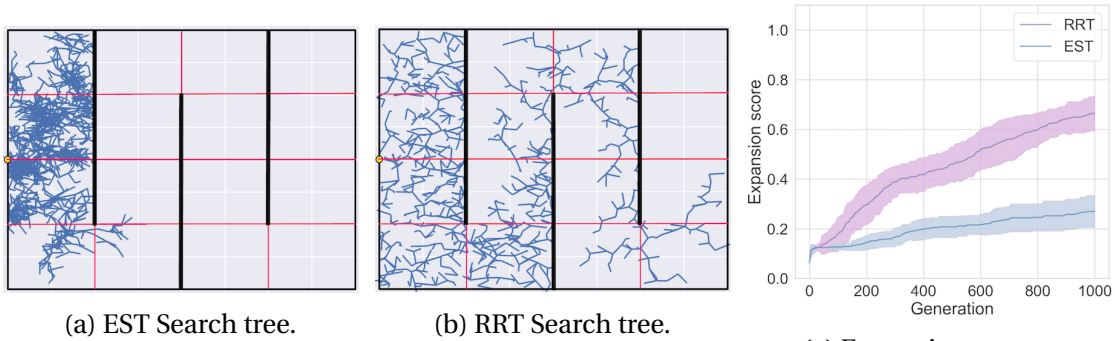
Algorithm 1 Rapidly-Exploring Random Trees

```
1: Initialize exploration tree:
2: T ← s_0
3: while iteration < N_sel/exp do
4:     s_samp ← random_config()                    ▷ selection operator
5:     s_sel ← nearest_neighbor(T, s_samp)
6:     u_rand ← random_control()                   ▷ random action
7:     s_new ← expand(s_sel, u_rand)               ▷ expansion operator
8:     T ← T.update((s_sel, s_new, u_rand))        ▷ update search tree
```

Algorithm 2 Expansive Spaces Trees (NS variant)

```
1: Initialize exploration tree:
2: T ← s_0
3: while iteration < N_sel/exp do
4:     N ← compute_weight(T)                       ▷ compute weights (novelty-like)
5:     s_sel ← select(T, N)                        ▷ selection operator (weight proportionate)
6:     u_rand ← random_control()                   ▷ random action
7:     s_new ← expand(s_sel, u_rand)               ▷ expansion operator
8:     T ← T.update((s_sel, s_new, u_rand))        ▷ update search tree
```

Comparative search properties of EST and RRT



(a) EST Search tree. (b) RRT Search tree. (c) Expansion scores.

Figure 6.2: Empirical comparison of EST and RRT. The SimpleMaze environment is divided into a 4×4 grid to compute the expansion scores of the MP algorithms. Search trees are shown after 1000 iterations. The means and standard deviations of the expansion scores are computed over 30 runs.

We empirically compare the exploration properties of the selection operators of EST and RRT in the “SimpleMaze" environment which is further described in Section 6.2.1. The pseudo-codes of both algorithms are given in Algorithms 1 and 2.

To assess expansion, we divide the maze into a 4×4 expansion grid shown in Figure 6.2. The expansion score is the number of zones containing at least one node over the total number of zones, i.e. 16.

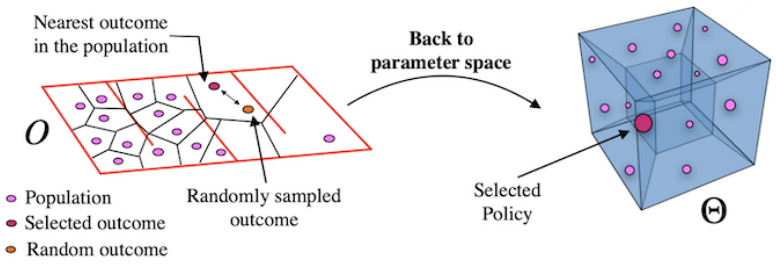
Both algorithms start with a single initial node in the middle of the left side. Figures 6.2a and 6.2b display exploration trees for both EST and RRT after 1000 iterations.

The evolution of expansion presented in Figure 6.2c shows that RRT explores the maze faster than EST.

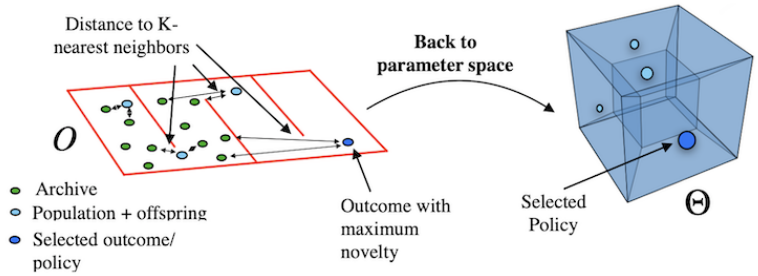
6.1.3 Application to diversity search algorithms

We now turn to the policy search context. In policy search, we consider a parametric policy π_θ where θ is a vector of parameters in a policy parameter space Θ.

As explained in Section 4, Diversity Search algorithms (DS) are policy search algorithms dedicated to covering a space of solutions as widely as possible. In particular, they can be used to find a target area without a reward signal. A common feature of these algorithms is that they define an *outcome space* $\mathcal{O}$ as a generally low-dimensional space that can characterize important properties of policy runs. The target area in such policy search problems is generally defined in $\mathcal{O}$. Thus it is natural to consider



(a) Goal-Exploration Process



(b) Novelty Search

Figure 6.3: Selection in GEP and NS. In GEP, an outcome is randomly sampled and the policy yielding the closest outcome is selected. In NS, the novelty is computed w.r.t to the archive. The policies yielding the most novel outcomes are selected.

that DS algorithms are performing a search in that space and to define the selection operator in that space.

But a key issue in the policy search context is that one cannot directly sample in $\mathcal{O}$, as the mapping from outcomes to policy parameters reaching these outcomes is generally unknown. As a consequence, search in these DS algorithms considers the mapping between Θ and $\mathcal{O}$, which we call the $f : \Theta \rightarrow \mathcal{O}$ mapping hereafter, see Figure 6.5.

Considering these two spaces results in crucial differences between MP and DS algorithms. In particular, while MP algorithms need to use a local expansion operator because they build a path to control a system from one configuration to another, DS algorithms rely on local expansions for different reasons.

Importantly, as it is not possible to sample directly in $\mathcal{O}$, the expansion operator must sample in Θ. Since selection operates in $\mathcal{O}$ and expansion in Θ but from the selected sample, one must determine the $\theta \in \Theta$ corresponding to the selected $o \in \mathcal{O}$. This problem is easily solved by storing in the archive a pair consisting of a θ and the resulting outcome o for each sample. For a selected o, a common approach for expansion is to simply apply a random mutation to the corresponding θ. For

the selection operator, the NS and GEP algorithms respectively implement the two strategies described in Section 6.1.1. Their pseudo-codes are given in Algorithms 3 and 4.

Algorithm 3 Vanilla Goal Exploration Process (Selection-Expansion variant)

Initialize population:
$P \leftarrow$ init_population()
while $generation < N_{generation}$ **do**
  for $i = 1 : N_{selection}$ **do**
    $o_{goal} \leftarrow random\_outcome()$ ▷ selection operator
    $(\theta_{sel}, o_{sel}) \leftarrow nearest\_neighbor(P, o_{goal})$
    $\theta_{new} \leftarrow expand(\theta_{sel})$ ▷ expansion operator
    $o_{new} \leftarrow evaluate(\theta_{new})$ ▷ compute outcome
    $P \leftarrow P + [(\theta_{new}, o_{new})]$

Algorithm 4 Novelty Search (Selection-Expansion variant)

Initialize population, expanded policies & archive:
$P \leftarrow$ init_population()
$M \leftarrow []$
$A \leftarrow []$
while $generation < N_{generation}$ **do**
  $N \leftarrow novelty(\{P + M\}, A + P)$ ▷ compute novelty scores
  $P' \leftarrow []$
  for $i = 1 : N_{selection}$ **do** ▷ population filtering
    $P' \leftarrow P' + [select(\{P + M\}, N)]$ ▷ selection operator (novelty proportionate)
  $P \leftarrow P'$
  $M \leftarrow []$
  for $(\theta_i, o_i)\ in\ P$ **do**
    $\theta_{new} \leftarrow expand(\{\theta_i\})$ ▷ expansion operator
    $o_{new} \leftarrow evaluate(\theta_{new})$ ▷ compute outcome
    $M \leftarrow M + [(\theta_{new}, o_{new})]$
  $A \leftarrow A + sample(M, N_{filter\ archive})$ ▷ archive filtering

Selection in NS

Novelty Search considers two sets of points in $\mathcal{O}$: the population and the archive. Only the policies contained in the population may be selected. We explain later how these sets of points are constructed.

Selection in NS can be performed using various selection operators. The uniform selection operator, the score proportionate selection operator, and the tournament-based operators are the most common ones (Cully and Demiris, 2018b). In this thesis,

we focus on the score proportionate selection operator biased toward more novel policies.

The idea behind score proportionate selection is to construct a probability distribution according to the **novelty scores** of the policies contained in the population. The novelty score $\mathcal{N}$ of a point $o \in \mathcal{O}$ is defined as the average distance to the k-nearest neighbors $(\mu_1, \ldots, \mu_k) \in \mathcal{O}^k$ in the archive, k being a hyper-parameter:

$$\mathcal{N} = \frac{1}{k} \sum_{i=1}^{k} \texttt{dist}(o, \mu_i). \tag{6.4}$$

Given a population $\{(\theta_i, o_i)\}_{i \in \{1,\ldots,N\}}$ containing N policies, the probability p_k for policy θ_k to be selected is proportional to its novelty score:

$$p_k = \frac{\mathcal{N}_k}{\sum_{i=0}^{N} \mathcal{N}_i}. \tag{6.5}$$

This is an instance of Strategy 1 described in Section 6.1.1 where the distance to neighbors is computed through the novelty score.

Selection in GEP

The selection operator in GEP works as follows. First, the agent draws a random target outcome o_{goal}. The agent would like to find a set of policy parameters θ_{goal} producing o_{goal}. For that, it looks in the archive for the closest outcome o_{sel} to o_{goal}, and it selects the policy parameters θ_{sel} which generated o_{sel}. This is clearly an instance of Strategy 2.

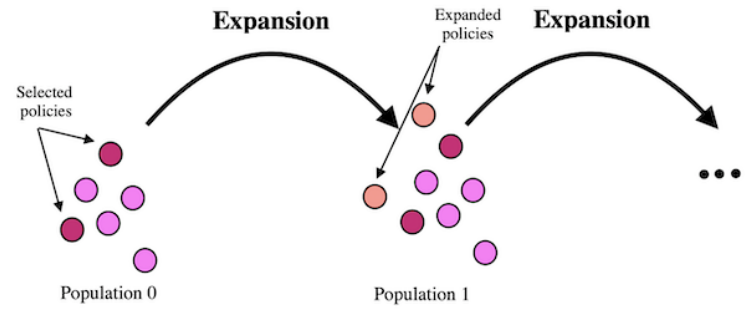
Since the GEP selection operator draws a random outcome and selects a policy corresponding to the closest outcome in the archive $\{(\theta_i, o_i)\}_{i \in \{1,\ldots,N\}}$, the probability p_k for policy θ_k contained in the archive to be selected is proportional to the volume of the Voronoi cell of its outcome o_k, as explained for RRT in Section 6.1.2:

$$p_k = \frac{volume(Voronoi\ cell\ o_k)}{\sum_{i=0}^{N} volume(Voronoi\ cell\ o_i)}. \tag{6.6}$$

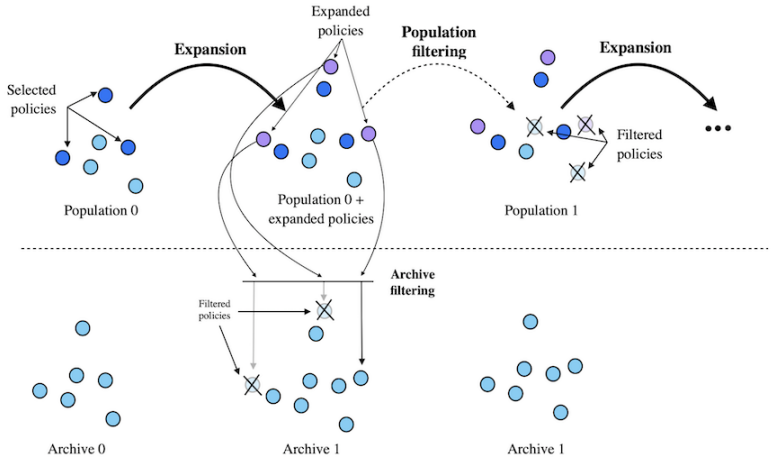
One can immediately see that the selection operator is exactly the same as in RRT, but acting in a different space.

Filtering in NS

In addition to their selection operators, NS also differs from GEP by using two filtering mechanisms.



(a) Goal-Exploration Process.



(b) Novelty Search.

Figure 6.4: Expansions in GEP and NS. In GEP, all expanded policies are added to the population. In NS, two filtering mechanisms are applied to the population and the archive.

Filtering the population The notion of population differs in GEP and NS. In GEP, the population gathers all policies since the first generation (see Figure 6.4a). At each iteration, all expanded policies are added to the population. In NS, the population is composed of a fixed-size set of policies updated at each generation. As in GEP, it is initialized with random policies. However, after expanding the policies contained in the population, only the most novel policies contained in the set { new population + new offspring } are selected to construct the new population. This approach encourages the policies to move in the outcome space from one generation to another and thus

promotes exploration (Doncieux et al., 2020).

Filtering the archive Beyond the population, NS uses another set called the archive to keep track of the policies evolved in past generations. The archive is initialized with the random policies used to initialize the population. At each generation, after expanding the population, about 10% policies are randomly sampled among the offspring and added to the archive, to keep the archive small. Indeed, the archive is used to compute the novelty score. Keeping it small limits the cost of finding the k-nearest neighbors.

The archive in NS is only used to compute the novelty score of policies contained in the {population + offspring} set. Policies from the archive are not added to the new population. If a policy contained in the { population + offspring } set is not selected for expansion, it is discarded and lost for future generations.

6.1.4 Similarities between MP and DS algorithms

It should now be obvious that, if we consider their most local expansion operators, NS shares similarities with EST and GEP with RRT. Indeed, the selection and expansion operators of the DS algorithms are closely related to the same operators of their MP counterparts.

Selection

From the side of NS and EST, their selection operators measure how isolated a sample is by attributing weight to each sample proportional to the inverse of the density of the archive in its neighborhood. The variants of EST and NS considered in this chapter use the same weight computation based on the mean distance of samples to their k-nearest neighbor in the exploration tree (see (6.1)) or the archive (see (6.4)). Therefore, nodes in EST and policies in NS share the same selection probability (see (6.2) and (6.5)) in different spaces.

Similarly, GEP and RRT also use a similar selection operator based on the volume of the Voronoi cells of the nodes/outcomes (see (6.3) and (6.6)) in their respective space.

Expansion

The expansion operator used in MP highly depends on how much we can steer a system in the desired direction. But, in the unknown system case, a standard strategy

consists in applying a single random control. This strategy relies on a strong assumption about the dynamical system. To ensure that the algorithm is guaranteed to find a path between the starting configuration and the goal configuration given an infinite amount of selection-expansion iterations (i.e. the algorithm is probabilistic complete (LaValle, 2006)), the system is assumed to be Lipschitz-continuous (Kleinbort et al., 2018). This assumption means that with enough expansions from the same node, a node should finally expand in the right direction.

In DS algorithms, the standard expansion operator applies a random perturbation to the selected policy parameters, which has similarities with the use of random actions for local expansions in the MP context.

However, reasons for using a local expansion operator are different in the MP and DS contexts. In the MP context, one needs to locally control the system along a path from the current configuration to the target configuration. In GEP, a local random perturbation is applied to the selected policy hoping that, the corresponding outcome being close to the sampled goal, the perturbed policy produces an outcome that is also close (and possibly closer) to this goal. One can see that the application of this selection-expansion strategy relies on the assumption of a smoothness property in the $f : \Theta \rightarrow \mathcal{O}$ mapping, i.e. that similar parameters yield similar outcomes. In the case of NS, the reason for using a local expansion operator is less straightforward. It relies on the assumption that, if a policy resulted in an outcome in a low-density region, a perturbed version of the policy should also result in an outcome exploring this low-density region, and thus be potentially helpful in the search for new outcomes. Again, this is equivalent to assuming a smoothness property in the f mapping.

6.1.5 Expansions in DS are often non-local

Even though DS have good reasons to use local expansions just like MP, expansions in DS are often non-local. Indeed, different sources of non-locality can be identified by dissecting the $f : \Theta \rightarrow \mathcal{O}$ mapping and considering a policy space Π and a trajectory space T (see Figure 6.5 for the details of the sub-mappings).

The first source of non-locality originates from the m_Θ mapping relating Θ to policies, in particular when they are modeled as non-linear neural networks. Even though Multi-Layers Perceptrons (MLPs) are continuous functions, if the magnitude of the perturbation is too large, the expanded version of a policy may yield a very different policy. Figure 6.6a illustrates the consequences of a random mutation.

The second source of non-locality lies in the nature of outcomes, which, as mentioned

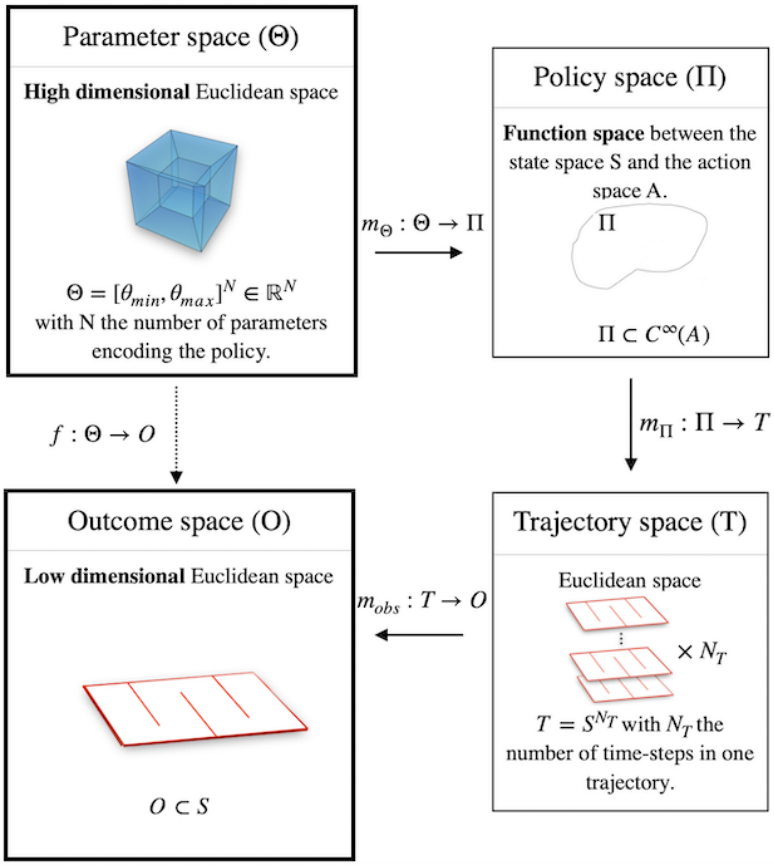
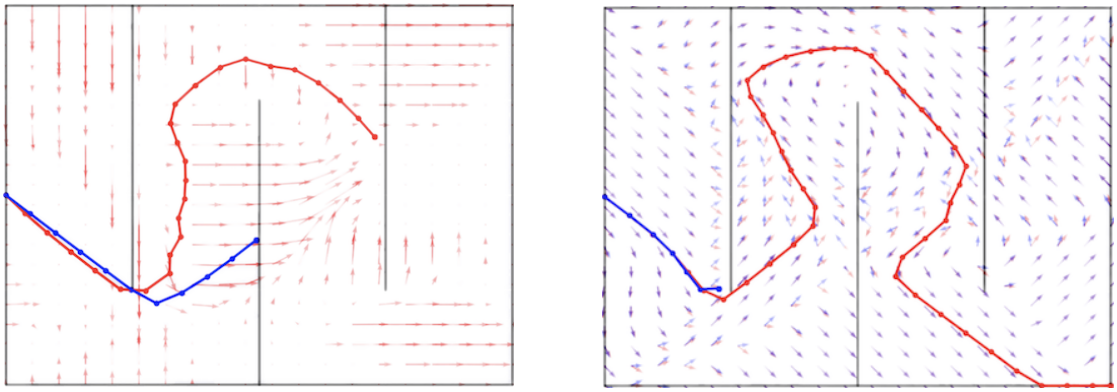


Figure 6.5: Description of the different spaces and mappings composing the $\Theta \to \mathcal{O}$ mapping in diversity search algorithms. Two intermediate spaces are considered: a policy space Π and a trajectory space T. Three sub-mappings m_Θ, m_Π and m_{obs} are also considered such that $f = m_\Theta \circ m_\Pi \circ m_{obs}$.

in Section 6.1.3, depend on policy runs, and therefore on trajectories. After selecting a pair $(\theta_{sel}, o_{sel}) \in \Theta \times O$, the expansion operator in DS perturbs θ_{sel} to obtain a new policy with parameters $\theta_{new} = \theta_{sel} + \delta\theta$ with $\delta\theta$ sampled from a spherical Gaussian distribution (Such et al., 2017) or a more complex distribution (K. Deb and D. Deb, 2014). The new policy $\pi_{\theta_{new}}$ yields trajectories defined by the equation

$$s(T) = s_0 + \int_0^T \mathcal{D}_{sys+env}(s(t), \pi_{\theta_{new}}(s(t))dt \tag{6.7}$$

which integrates the dynamical system $\dot{s} = \mathcal{D}_{sys+env}(s, \pi_{\theta_{new}}(s))$ over the time interval $[0, T]$, where $\mathcal{D}_{sys+env}$ models the dynamics of the system in interaction with its environment and s_0 is the starting state of the rollout. Even if the magnitude of the mutation is kept low enough for the expanded policies to be very close to the selected one, the numerous time steps of control may result in a large deviation between the



(a) (b)

Figure 6.6: (a) Lack of smoothness caused by the non-linearity of the neural network policy. The difference between two policies π_θ and $\pi_{\theta+\epsilon}$ is visualized as a vector field (in red). π_θ yields the red trajectory, and $\pi_{\theta+\epsilon}$ yields the blue trajectory. Although the parameters are close in Θ, output differences are accumulated and lead to very different trajectories. (b) Lack of smoothness caused by the environment. The non-linearity or discontinuity of the environment (here at the extremity of the first wall) can cause similar actions to have dramatically different effects. This can lead to huge outcome differences for similar policy parameters.

trajectories obtained by the two policies via Equation (6.7) as differences accumulate over time steps. These errors may be aggravated by discontinuous dynamical systems or environments $\mathcal{D}_{sys+env}$ and result in a non-smooth mapping m_Π from policies to trajectories, and therefore from policies to outcomes. For instance, in maze environments, two close policies may yield very different trajectories if one trajectory gets blocked by a wall, as illustrated in Figure 6.6b.

Preliminary conclusion: performance assumptions

The similarities outlined above suggest that, if the expansion operators have similar properties, NS and GEP should share exploration abilities that are similar to those of EST and RRT respectively. However, for common $f : \Theta \to \mathcal{O}$ mappings, small perturbations in Θ may result in large changes in $\mathcal{O}$, and this lack of smoothness can result in a two situations.

- If the lack of smoothness of the $f : \Theta \to \mathcal{O}$ mapping is too serious, one could hypothesize that the use of local expansions in DS should bring no advantage compared to a random sampling of policy parameters.
- Or, the $f : \Theta \to \mathcal{O}$ mapping could be smooth enough to let NS and GEP both

outperform random sampling, but not smooth enough to inherit the search properties of EST and RRT.

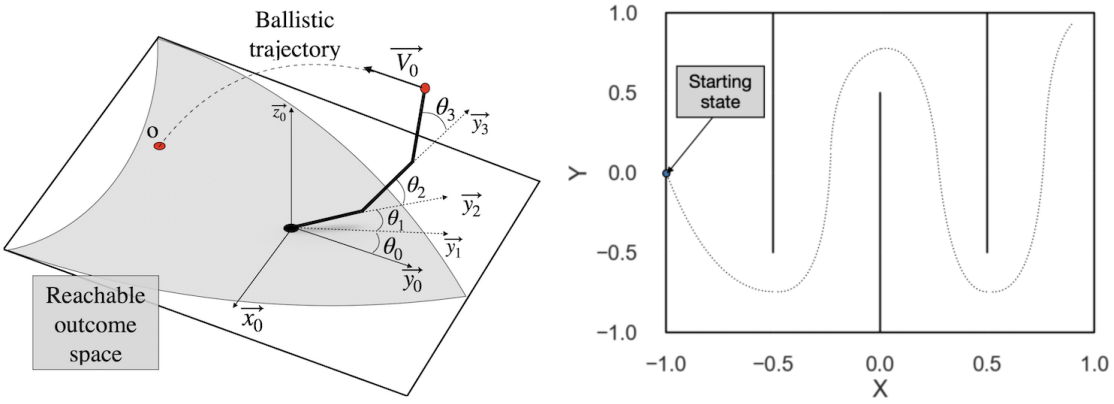
Below, we investigate these two possibilities experimentally.

6.2 Experimental study

In this section, we experimentally study NS, GEP and a random search baseline using two environments with different smoothness properties to assess whether NS and GEP inherit from the properties of EST and RRT.

6.2.1 Experimental setup

The experimental comparison is based on two environments: a ballistic task using a 4-DOF simulated robot arm and a maze environment called SimpleMaze, see Figure 6.7. In both environments, the state space is continuous and time is discrete.



(a) 3D ballistic throw.

(b) SimpleMaze.

Figure 6.7: Studied environments. (a) In 3D ballistic throw, an agent controls the angular speed $(\dot{\theta}_i)_{i\in[0,3]}$ of a 4-joint 3D robot arm in order to throw a projectile. The outcome o of the policy is the final position of the projectile. (b) In SimpleMaze, the agents start from a fixed position $(-1,0)$ and must reach the upper right corner.

3D ballistic throw

The planar robot arm ballistic throw environment simulates the trajectory of a projectile thrown by a 3D 4-joint robot arm inspired from (Cully, 2019a). The velocities $(\dot{\theta}_i)_{i\in[0,3]} \in [-1rad.s^{-1}, 1rad.s^{-1}]$ of the joints of the robot arm are controlled by an

MLP. The throw is divided into acceleration-release phases. The acceleration phase is a single time step of control of the robot joints. After the acceleration phase, the end effector of the robot releases the projectile which then follows a ballistic trajectory. The outcome is the (x, y) coordinates of impact.

The expansion operator of DS algorithms truly achieves local expansions in this environment. First, by decreasing the magnitude of the polynomial mutations with $\eta = 2000$ (K. Deb and D. Deb, 2014), we make sure that any mutated policy is similar to the one from which it originated. Second, by reducing the acceleration phase to a single time step, we avoid the accumulation of differences between trajectories of the dynamical system controlled by the mutated policy and by its parent. This ensures the smoothness of the m_Θ mapping. Moreover, without obstacles, there is no discontinuity in the trajectory of the projectile, and the function $\mathcal{D}_{sys+env}$ (Equation (6.7)) remains smooth under all circumstances. Under these constraints, the $f : \Theta \to \mathcal{O}$ mapping is such that small perturbations in Θ yield small changes in $\mathcal{O}$.

SimpleMaze

We chose a Maze environment as it facilitates visualization of the exploration properties of the algorithms. A rollout lasts 50 time steps. The outcome corresponds to the final position of the agent at the end of the rollout. The agent starts from $(-1, 0)$ and receives at each time step the position of $(x, y) \in [-1, 1] \times [-1, 1]$ at the current time step as input and outputs the next displacement $(dx, dy) \in [-0.1, 0.1] \times [-0.1, 0.1]$. The agent does not perceive the walls. The magnitude of the polynomial mutations ($\eta = 15$), the duration of a rollout as well as the discontinuities caused by the walls result in non-local expansions (as shown in Figure 6.6), with similar policy parameters leading possibly to very different outcomes.

6.2.2 Metrics

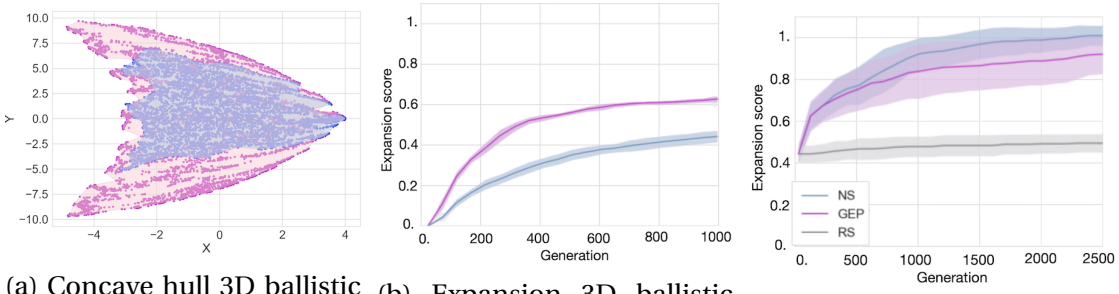
In order to assess the expansion of a DS algorithm, we use the previously defined domain expansion metric. We split $\mathcal{O}$ into a $G_{expansion} \times G_{expansion}$ expansion grid. The expansion score corresponds to the number of expansion cells filled with at least one policy over the total number of expansion cells.

6.2.3 Implementation details

In both GEP and NS, a KD-tree (Bentley, 1975a) is used to accelerate the nearest-neighbor computations. New policies are added at each generation. NS keeps the cost

of reconstructing the KD-tree low via the filtering mechanism of its archive. However, in the vanilla version of GEP, every expanded policy is added to the tree. As a result, the size of the tree surges and reconstructing the KD-tree becomes computationally very expensive. To avoid reconstructing the tree at each iteration, a second small KD-tree keeps track of the most recent policies and is reconstructed at every generation. The main KD-tree is updated only every N_{update} after transferring the content of the small KD-tree. The hyper-parameters are summarized in Chenu et al., 2021.

6.3 Results



(a) Concave hull 3D ballistic throw.

(b) Expansion 3D ballistic throw.

(c) Expansion SimpleMaze.

Figure 6.8: (a) Visualisation of the concave hull of the exploration trees of NS (blue) and GEP (pink) in 3D ballistic throw. (b) Expansion scores of GEP and NS in 3D ballistic throw. As the expansion operator is local, GEP inherits exploration properties from RRT and expands faster than NS. (c) Expansion scores of GEP, NS and a random search (RS) baseline in SimpleMaze. The non-local expansions result in NS exploring slightly faster than GEP. Both NS and GEP outperform RS.

6.3.1 Results on 3D ballistic throw

In this section, we verify that in this environment GEP and NS inherit the search properties of RRT and EST, and that, as expected, GEP explores $\mathcal{O}$ faster than NS.

Figure 6.8b presents the expansion score obtained by GEP and NS after 1000 generations. It confirms that GEP expands faster across $\mathcal{O}$ and converges quickly towards a maximum[I] that NS struggles to reach. These results are analogous to the performance of RRT and EST described above. The similarities between the selection operator of both DS algorithms and their MP counterpart enables similar exploration performance in the ballistic task, an environment that preserves the locality of expansions.

[I]The non-rectangular shape of $\mathcal{O}$ in the 3D ballistic throw environment makes some cells of the expansion grid unreachable, which explains why GEP eventually covers only about ~ 60% of $\mathcal{O}$.

6.3.2 Coverage visualization

Figure 6.8a presents the concave hull obtained by the exploration trees of GEP and NS after 500 generations in one run. We observe that NS spends numerous generations paving the center of the reachable outcome space ($|Y| < 5$) whereas GEP expands in all directions and quickly finds the limits of the reachable search space.

6.3.3 Results in SimpleMaze

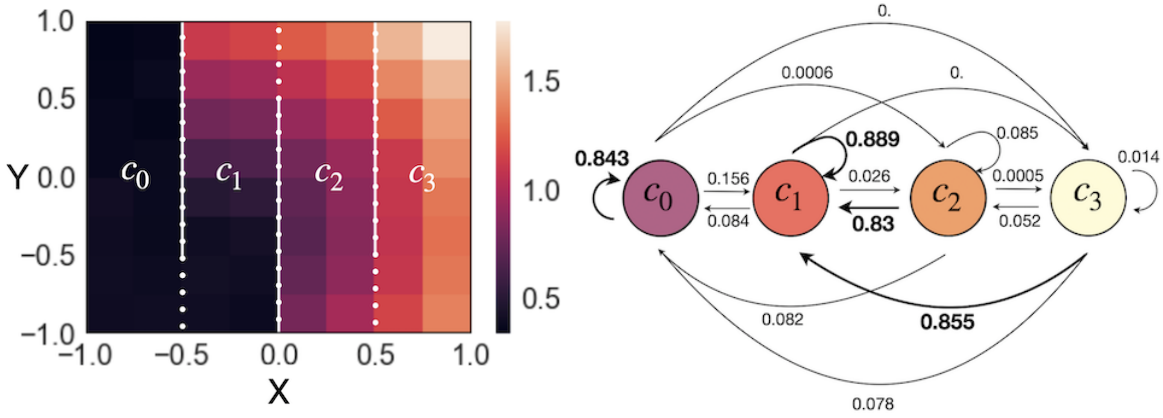


Figure 6.9: Quantitative illustration of non-local expansions. (Left) Mean distance between the outcome of a selected policy and the outcomes of its expanded versions depending on its position in the SimpleMaze. We observe that this distance increases as we progress through the maze i.e. the expansion operator gets increasingly more non-local. The mean distances are computed for 100 expansions per selected policy and 200 selected policies per cell. Results are averaged over 10 runs of NS and 10 runs of GEP. (Right) Markov Chain representation of the expansion in the outcome space using four subsets of outcomes. Each subset corresponds to a corridor in the SimpleMaze. The empirical transition probabilities are computed for 100 expansions per selected policy and 200 selected policies per cell. We observe that most mutations in the second, third and fourth corridors yield an outcome in the second corridor. There is a low probability of exploring the next corridor (or the same) when expanding a policy that reaches an advanced corridor.

We previously argued that, if the expansion operator is local, NS and GEP have search properties that are similar to their MP counterparts. We also explained that the $f : \Theta \rightarrow \mathcal{O}$ mapping is not smooth in SimpleMaze, which results in non-local expansions. In this section, we assess the consequences of non-local expansions by comparing the exploration performance of NS and GEP. We show that this non-locality results in losing the properties inherited from RRT and EST.

Figure 6.8c presents the expansion and density scores obtained by NS, GEP and RS in Simple Maze. It shows that both NS and GEP outperform RS. Therefore, the $f : \Theta \mapsto O$ mapping is smooth enough for NS and GEP to benefit from their selection operator.

However, the performances of NS and GEP in SimpleMaze differ from their performances in the ballistic throw. NS and GEP perform similarly during the first 500 generations. Then, the expansion of NS accelerates and outperforms the expansion of GEP by achieving a full exploration of the maze after about 2500 generations while GEP generally fails to reach the end of the maze and only reaches an expansion score of 0.9 after the same number of generations.

The difference in expansion rates after 500 generations arises from a progressive degradation of the locality of the expansion operator. As explained in Section 6.1.5, because of the non-locality caused by the discontinuous environment due to walls and the non-linear neural network policy, expanding a policy that yields an outcome in the second corridor or beyond often results in a policy blocked by the first wall. Figure 6.9 illustrates this degradation by underlining the increasing distance between the outcome of a selected policy and the outcome of its expanded versions as the selected policy progresses through the maze. We see that the further we progress through the maze, the more distant the outcome of an expanded policy gets from the outcome of a selected policy. In other words, the better the policy, the less local the expansion operator becomes. In addition, the Markov Chain representation of the expansions shows how non-local expansions for policy advanced in the maze mostly result in perturbed policies blocked in the second corridor i.e. policies blocked by the first and second walls.

However, these better policies constitute promising stepping stones for further exploration (Woolley and Stanley, 2012). Therefore, it is important to select the most advanced policies to discover policies yielding outcomes further in the maze eventually. That is exactly what the filtering mechanisms of NS do. Figure 6.10 presents the outcomes of the selected policies by NS and GEP through one run. In the left panel of Figure 6.10, the smooth change of colors shows that the selected policies by NS at a given generation correspond to the most advanced policies in the maze.

On the contrary, GEP does not integrate any filtering mechanisms. Therefore, GEP keeps sampling policies blocked in the already well-explored areas of the maze (see right panel of Figure 6.10). Thus, GEP requires more generations to deal with the degraded expansion operator which results in a slower increase of the expansion score.

These results show that, in an environment where the $f : \Theta \rightarrow \mathcal{O}$ mapping lacks

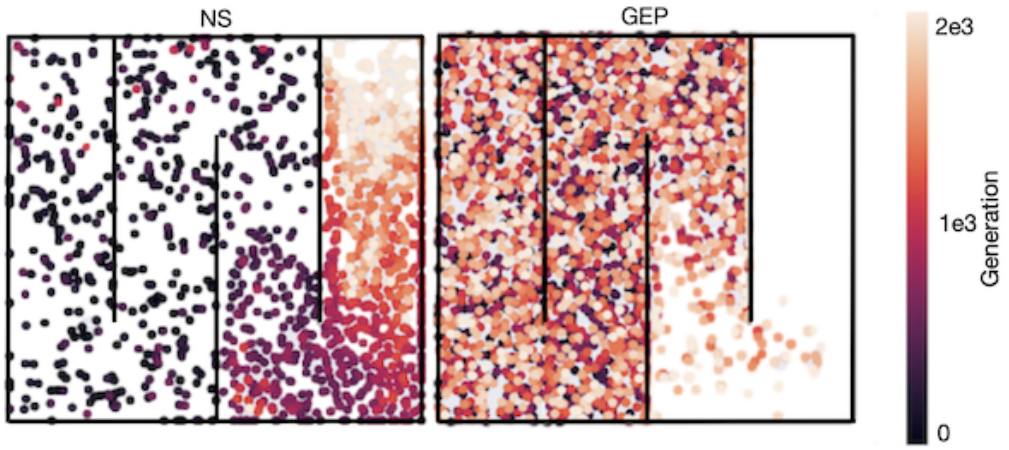


Figure 6.10: History of the selected policies in SimpleMaze (2000 generations run). Using its filtering mechanism, NS keeps selecting the most advanced policies in the maze. On the contrary, GEP keeps sampling policies blocked in the already well-explored areas of the maze.

smoothness, the properties inherited from RRT and EST are lost, which means that the selection-expansion mechanisms do not behave as originally intended. Even though both NS and GEP outperform random sampling of policies, additional heuristics such as filtering mechanisms must be exploited to overcome difficult expansions with degraded expansion operators.

6.4 Conclusion

In this Section, we presented a comparison between two diversity search algorithms: GEP and NS. We started by presenting a unifying framework called selection-expansion which draws a parallel between both algorithms and two Motion Planning algorithms, EST and RRT.

An experimental study showed that in an environment like the 3D ballistic throw where the $f : \Theta \rightarrow \mathcal{O}$ mapping is smooth enough, GEP and NS inherit the exploration properties from their Motion Planning counterpart. In that case, GEP explores faster the environment than NS.

By contrast, maze results show that, even though GEP and NS share common selection-expansion properties with RRT and EST, they do not share the same exploration abilities if the expansion operator is not local. In such situations, the experimental study showed that NS outperforms GEP by using efficient filtering mechanisms.

This work opens up the question of restoring locality in complex environments where

the expansion operator is non-local.

One could consider several alternatives to restore the locality partially. In (Lehman et al., 2018), the authors use safe mutations to constrain the perturbation of the parameter. Mutation parameters are scaled according to their impact on the output of the policy. The impact is computed using first-order or second-order gradient-based information.

A different parameterization of the controllers and a better outcome space design can also help restore locality at the cost of leaving the context of neuro-evolution and losing representation capabilities. In (Huber et al., 2022; Morel et al., 2022), the authors use a concatenation of waypoints corresponding to successive configuration of the robot as the outcome space and a combination of a polynomial interpolation and a Proportional Derivative controller to guide the robot between them. As a result, a small change of in the waypoints should result in a small trajectory change.

However, these two alternatives only tackle one source of non-localities coming from the non-linear nature of the controller. They are agnostic to the discontinuities and non-linearities of the environment (e.g. the walls in SimpleMaze) which are still a major source of non-locality in the expansion operators. In the next section, we adopt a Divide & Conquer strategy and propose a skill-chaining based approach to bypass this second source of non-locality.

7 Novelty Search Skill-Chaining: adopting a Divive & Conquer strategy

In Chapter 6, we showed that expansions in Diversity Search (DS) algorithm are likely to be non-local because of the discontinuities and non-linearities contained in the mapping $f : \theta \rightarrow \mathcal{O}$ between the parameter space and the outcome space (e.g. non-linear controllers, environments with discontinuous dynamics, etc.).

In this Chapter, we tackle this issue by adopting a Divide & Conquer strategy and we introduce Novelty Search Skill Chaining (NSSC). Unlike NS which searches for a single policy, NSSC constructs a skill chain to restore the locality of the expansion operator of DS algorithms in complex environments. Instead of starting to explore from the reset state of the environment, NSSC uses a skill chain to return to an advanced state and start exploring from there. Exploration from advanced reset states allows the agent to avoid all the potential sources of non-locality between the reset state of the environment and the advanced reset state (see Section 7.3.2).

In Section 7.3.3, we discuss the similarities between NSSC and sampling-based MP algorithms. Building on this comparison, we underline the potential limits of NSSC in non-holonomic environments (see Section 7.3.4).

In an experimental study, we first evaluate NSSC in the Simple Maze environment to show that NSSC restores locality in the expansion operator of DS algorithms. We then evaluate NSSC in a non-holonomic version of the Simple Maze environment to underline the limitations of a forward construction of a skill chain guided by low-dimensional outcomes.

7.1 Related Work

NSSC uses skill-chaining to return to advanced states in order to explore locally. In this section, we cover the literature on Skill chaining and exploration from advanced states.

7.1.1 Skill-chaining

Skill-chaining (Konidaris and Barto, 2009; Konidaris et al., 2010) is an example of the application of the Divide & Conquer paradigm in RL. Instead of solving a complex RL problem using a single policy, skill-chaining learns a set of policies on simpler tasks and chains them together to solve the complete problem.

For example, this principle is applied in the Play-Backplay-Chain-Skill (PBCS) algorithm (Matheron et al., 2020). In PBCS, the Backplay algorithm (Salimans and R. Chen, 2018) is used to learn a set of skills backward from the final state of a single demonstration obtained using a planning algorithm.

In (Bagaria and Konidaris, 2019; Bagaria, Senthil, Slivinski, et al., 2021), skills are formalized using the options framework ((Sutton et al., 1999; Precup, 2000)). An option is composed of an initial set of states in which this option can be activated, a termination function that decides if the option should be terminated given the current state, and the intra-option policy which controls the agent at a single time-step scale to execute the option. After completing an option, the agent uses an inter-option policy to decide which option should be applied depending on its current state. In Deep Skill Chaining (DSC) (Bagaria and Konidaris, 2019), for a given goal, a first option is learned to reach it from a nearby region reliably. Then, iteratively, a chain of options is created to reach the goal from further states. The goal of each option is to trigger the initiation condition of the next one, and the phase of construction of the initiation classifiers requires various successful runs randomly obtained via exploration or RL.

Both PBCS and DSC construct the skill chain backward from the desired outcome. On the contrary, other methods (including ours) construct the skill chain in a forward manner.

In Hierarchical Behavioral Repertoire (HBR) (Cully and Demiris, 2018a), increasing complexity skills are constructed sequentially to obtain complex movements with a planar robotic arm. HBR starts by learning low-level skills corresponding to short trajectories. Then, it chains those skills to construct more complex trajectories. Therefore, the learned complex movement constitutes a skill chain. HBR was recently

extended in the Hierarchical Trial and Error algorithm for efficient damage recovery with a hexapod robot Allard et al., 2022.

In Deep Skill Graphs (Bagaria, Senthil, and Konidaris, 2021), the authors construct a graph of skills using an RRT-like strategy directly in the goal space of a goal-conditioned RL agent (see Section 3). Each branch in the graph constitutes a chain of skill and is extended forward using a learned model and Model Predictive Control Garcia et al., 1989.

Such a forward construction of skill chain may seem advantageous as it does not require a demonstration, as in PBCS, or an easy-to-explore environment as in DSC. However, depending on the properties of the environment, this strategy can have critical limitations (see Section 7.3.4).

7.1.2 Returning to advanced states to explore

Going back to advanced states to explore the state space locally is the central idea of several algorithms. In the Go-Explore algorithm (Ecoffet et al., 2019b), the outcome space is divided into cells. During an iteration of the exploration phase, the agent returns to previously visited states contained in rarely visited cells and explores locally using random actions. In order to return to previously visited states, Go-Explore assumes that the agent can be reset to a previously visited state or, that the environment is deterministic such that the agent can return to a previously visited state using the same sequence of actions.

In First Return, Then Explore, a more recent version of the Go-Explore algorithm (Ecoffet et al., 2021), the agent uses a Goal-Conditioned Policy to return to previously visited states. This version of the algorithm achieved state-of-the-art results in complex Atari games that require the agent to explore extensively to master the game. The limited application of this approach in a robotics environment is discussed in Chapter 8.

In the UPSIDE algorithm (Kamienny et al., 2021), the agent follows a sequence of policies to return to advanced states and then enters a diffusion phase. During the diffusion phase, the agent explores the state space locally using intrinsic motivation based on *empowerment* (see Chapter 3.3.2). More precisely, exploration is encouraged via Mutual-Information skill discovery (Gregor et al., 2016; Eysenbach et al., 2018). The sequence of policies used to return to advanced states can be seen as a skill chain.

7.2 Background

7.2.1 Invalid Exploration States

In a DS algorithm, the behavior of a policy is characterized by a low-dimensional outcome. Often, this outcome corresponds to the projection of the last state of the trajectory in a low dimensional subspace corresponding to the DoFs that the agent must explore. (e.g. the (x, y) positions in a 2D maze of the Chapter 6). Therefore, if the state space is high-dimensional, the same outcome may correspond to very different final states.

This can be problematic if the agent needs to start exploring from this final state of the trajectory. Indeed, it is possible that this final state corresponds to configurations from which exploration is impossible, as in MP (see Section 3). Such states from which the agent cannot explore are very common in non-holonomic environments like complex robotics systems. Indeed, in a non-holonomic environment, connecting two close configurations may require long trajectories in the configuration space if the shortest path is not admissible. Even worse, connecting two close configurations may be impossible (see Section 2).

Therefore, if the final state contains a configuration that cannot be connected to any further configuration helping the exploration of the state space, exploring from this state is fruitless. We refer to these states as *invalid exploration states*.

7.3 Method

In this Section, we introduce the Novelty Search Skill Chaining (NSSC) algorithm. First, we present a key assumption made by most DS algorithms and which is even more crucial in NSSC. We then explain how NSSC, by using skill-chaining, tackles the problem of non-local expansions. Finally, we underline the limits of NSSC in non-holonomic environments.

7.3.1 Deterministic Environments

Most DS algorithms require that the environment is deterministic (Justesen et al., 2019; Flageat and Cully, 2020; Chalumeau et al., 2022). Indeed, as discussed in Sections 4.3.3 and 6.1.3, a policy is characterized by an outcome that depends on the trajectory resulting from its evaluation. In a non-deterministic environment, running twice the same policy may result in different trajectories and, therefore, different outcomes.

This could interfere with the selection process. For instance, suppose a sub-optimal policy is considered highly novel because of a noisy evaluation. In that case, this policy has a large chance of being selected for multiple expansions that will probably result in no additional exploration. In NSSC, the use of skill-chaining to return to advanced states makes this assumption even more crucial.

7.3.2 Skill-Chaining allows expansions to be local in the outcome space

In NSSC, the population is initialized with randomly initialized policies, and the expansion operator perturbs a policy to obtain a new one, just as in NS. Assuming that the environment is deterministic, the agent uses the original policy to return to the final state of its evaluation trajectory. From this state, it extends the trajectory using the new policy.

Returning to this final state allows the agent to avoid all the potential non-smooth components of the environment dynamics that the original policy has already passed (e.g. the first walls in the Simple Maze). Therefore, the agent can explore the state space locally around the advanced state during the trajectory extension. The total trajectory resulting from the return to the advanced state and the exploration corresponds to the *skill-chaining* trajectory.

If the outcome resulting from the skill-chaining trajectory is novel enough, the *skill-chain*, composed of the original policy and the perturbated one, is kept in the population. Then, additional expansions of this skill chain may result in increasingly longer chains as more policies are added. If the skill chain contains more than a single skill, NSSC must decide from which skill the agent should explore locally. To do so, NSSC chooses the skill resulting in the most novel outcome. Thus, NSSC constructs a tree of skills spanning the state space rather than a single chain. The expansion step of NSSC is detailed in Algorithm 5. Note that when a skill chain is selected to be expanded, the expansion operator is applied to the selected skill.

7.3.3 Viewing NSSC as a sampling-based MP algorithm

Similarly to sampling-based MP algorithms (see Section 6.1.2), NSSC constructs a tree spanning the state space. Here, a node corresponds to the final state reached by a skill and an edge indicates that a skill connects two states. However, NSSC expands the exploration tree using a perturbed policy. This expansion operator results in a very different expansion dynamic of the tree, compared to sampling-based MP

Algorithm 5 Novelty Search Skill-Chaining expansion step

input $\{\pi_j\}_{j\in[1,N_{skills}]}, \{o_j\}_{j\in[1,N_{skills}]}$ ▷ Skill chain + skills outcomes
$\pi'_{N_{skills}} \leftarrow expand(\pi_{N_{skills}})$ ▷ Expansion in Θ
$L_{skills} \leftarrow \{\pi_j\}_{j\in[1,N_{skills}]} + \{\pi'_{N_{skills}}\}$ ▷ Extend skill chain
$s, d \leftarrow env.reset(), False$ ▷ Initialize agent to environment reset state
$\tau \leftarrow [s]$
$j_{\text{skill}} \leftarrow 1$ ▷ Initialize first skill
while $d == False$ **do** ▷ Return to advanced state + local exploration
    $t \leftarrow T$ ▷ Fixed budget for each skill
    while $t > 0$ **do**
        $a_t \leftarrow \pi_{j_{\text{skill}}}(s_t)$ ▷ Select actions using current skill
        $s_t, d \leftarrow env.step(a_t)$ ▷ Deterministic environment step
        $\tau \leftarrow \tau + [s_t]$
        if $d == True$ **then**
            $t \leftarrow 0$
        else
            $t \leftarrow t - 1$
    $j_{\text{skill}} \leftarrow j_{\text{skill}} + 1$ ▷ Increment skill
    if $j_{\text{skill}} > len(L_{skills})$ **then**
        $d == True$ ▷ Local exploration finished
output τ

algorithms (see Section 3.5). Indeed, usually in sampling-based MP, in the absence of a model to steer the system, a randomly sampled action is applied for a few steps ($\sim 5/10$) (LaValle, 2006). On the contrary, in NSSC, a policy is applied for longer control sequences (e.g. 50 steps in the *Simple Maze*) which should result in more complex behaviors. For instance, one can expect that the perturbed policy partly conserves knowledge about the dynamics that the parent policy captured. This conservation should result in more interesting sequences of transitions in the state spaces than randomly sampled actions. Ultimately, these longer control sequences should result in edges connecting more distant states in the state space.

Note that NSSC gets very close to Expansive Search Trees (EST) (David Hsu et al., 1998) if we reduce the number of control steps for each skill. Indeed, as discussed in Section 6.1.4, NS and EST already share a similar selection operator. In addition, expansions obtained using perturbated skills for a limited number of steps are very close to expansions obtained using a few random actions as in EST. However, whether expansions rely on long policy-based rollouts or short sequences resulting from randomly sampled action, exploration may be difficult depending on the environment.

7.3.4 NSSC in non-holonomic environments

The intuition behind returning to an advanced state using a skill chain is to explore the state space locally around the end of the skill chain. However, such expansion may be fruitless in a non-holonomic environment. Indeed, suppose the final state of the skill-chaining trajectory corresponds to an invalid exploration state (see Section 7.2). In that case, the agent cannot explore from this state. Returning to it to expand is useless.

For instance, if the point-like agent in the Simple Maze is replaced by a Dubins car (Dubins, 1957) (as in Section 7.4), a skill chain that collides with the wall cannot be extended any further as the agent is not allowed to drift or move backward. Expanding the exploration tree from this *invalid exploration state* results in the same state, i.e. no expansion at all.

One naive solution to this problem could be to redefine the outcome space to encourage diversity in the full configuration. Indeed, this would result in the exploration of different car orientations and, eventually, the discovery of *valid exploration states*. While this idea seems reasonable in the context of a 3-dimensional state space, it cannot be scaled to more complex systems with high-dimensional configuration spaces (e.g. humanoid robots). In that case, the outcome space should remain low-dimensional for efficient Diversity Search. However *invalid exploration states* may appear.

Although highlighted here in the context of skill-chaining, *invalid exploration states* may also be problematic for GCRL agents using low-dimensional goals (see Section 3.4). For instance, in Autotelic GCRL (Pong et al., 2019; Pitis et al., 2020; Castanet et al., 2022), if the curriculum of low-dimensional goals guides the agent towards *invalid exploration states*, the exploration of the goal space may get blocked. Most approaches are not confronted with this problem as they are evaluated in holonomic navigation environments similar to the *SimpleMaze*.

7.4 Experimental study

In this Section, we evaluate NSSC in two maze environments of different natures: the first environment is holonomic and the other is not. We show empirically that recovering a local expansion operator can benefit or harm NSSC depending on the environment.

7.4.1 Experimental Setup

The first environment is the Simple Maze introduced in Section 6.2.1. This environment is holonomic. Using skill-chaining, NSSC should recover a local expansion operator and, therefore, explore the state and outcome spaces locally.

The second environment is a non-holonomic variant of the Simple Maze. In this variant *Simple Dubins Maze*, the point-like agent is replaced by a Dubins Car (Dubins, 1957). The state $s = (x, y, \theta)$ includes the 2D position of the car in the maze and its orientation. The car only moves forward at a constant speed, and the agent controls only the change of orientation of the car.

The *Simple Dubins Maze* poses two main challenges. The first one is the non-holonomic constraints induced by the Dubins Car. Unlike in the *Simple Maze*, even in the absence of any wall, the agent cannot move in any direction of the state space. Indeed, the agent cannot move backward or drift. Therefore, if the agent collides with a wall, it gets stuck and can no longer explore. Those collision states correspond to *invalid exploration states*. Expanding a skill chain from those invalid exploration states results in no exploration.

To exacerbate this phenomenon, outcomes in the *Simple Dubins Maze* correspond to the final (x, y) position of the agent at the end of the skill chain (as in the *Simple Maze*). Therefore, the selection operator does not consider the orientation when selecting policies. There is no search for orientation diversity.

As discussed in the previous paragraph, searching for orientation diversity could eventually help bypass the non-holonomic constraints. However, although adding the orientation in the definition of the outcomes is an efficient solution in this environment, this approach cannot be applied to more complex environments with even higher state dimensions. For this reason, we keep the outcome space as small as possible, even in this toy problem.

Results in the *Simple Maze*

In this Section, we verify that using the skill-chaining strategy, NSSC recovers a local expansion operator.

Figure 6.9 presents the mean distance between the outcome of a policy and the outcome resulting from an expansion depending on the (x, y) position in the maze. We observe that, unlike with NS (see Figure 6.9) the distance is close to zero everywhere in the environment. In addition, the associated Markov Chain shows that an expansion is

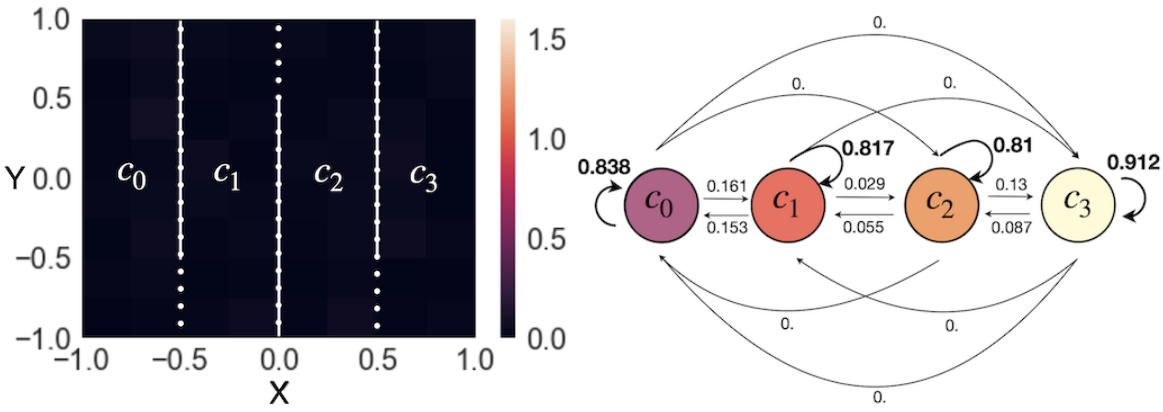


Figure 7.1: Quantitative evaluation of the locality of NSSC expansions in the *Simple Maze*. **Left:** Mean distance between the outcome of a selected policy and the outcomes of its expanded versions depends on its position in the SimpleMaze. We observe that this distance stays low in every area of the outcome space i.e. the expansion operator is local. The mean distances are computed for 100 expansions per selected policy and 200 selected policies per cell. **Right:** Markov Chain representation of the expansion in the outcome space using four subsets of outcomes. Each subset corresponds to a corridor in the SimpleMaze. The empirical transition probabilities are computed for 100 expansions per selected policy and 200 selected policies per cell. We observe that most expansions give a result in the same corridor. This results in a local exploration of the outcome space and, at some point, the exploration of the next corridor.

likely to result in another outcome contained in the same corridor. These results show that NSSC recovers a local expansion operator. However, does this local expansion operator allow NSSC to explore the outcome space faster?

Figure 7.2 presents the expansion score obtained by NS and NSSC after 500 iterations in the *Simple Maze*. NSSC requires less than 400 iterations to cover the maze fully while NS only covers ~ 70% of the maze after 500 iterations. This result confirms that NSSC benefits from the restoration of the locality of the expansion operator in a holonomic environment.

Results in the *Simple Dubins Maze*

In the *Simple Dubins Maze*, when the agent is replaced by a Dubins Car and the environment becomes non-holonomic, the local expansion operator penalizes NSSC.

Figure 7.2 presents the expansion score obtained by NS and NSSC after 500 iterations in the *Simple Dubins Maze*. While the natural forward dynamic of the Dubins Car helps

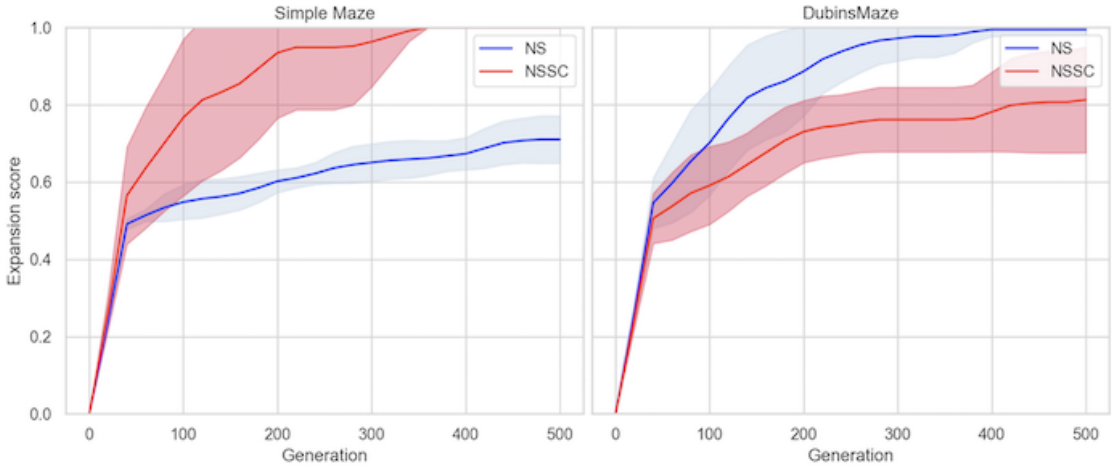


Figure 7.2: NSSC / NS comparison in the Simple Maze and the Simple Dubins Maze. In the Simple Maze, NSSC benefits from the skill-chaining strategy and can explore the outcome space locally, resulting in a faster exploration. On the contrary, in the Simple Dubins Maze, the skill-chaining often results in invalid starting states. Indeed, the walls block many expansions in NSSC. As a result, NSSC explores slower than NS.

NS to explore the *Simple Dubins Maze* faster than the *Simple Maze*, NSSC struggles to achieve total coverage of the maze. Indeed, after 500 generations, NSSC only covers 80% of the environment. This result illustrates how NSSC is penalized by returning to *invalid exploration state* to explore locally. Indeed, each expansion from these states

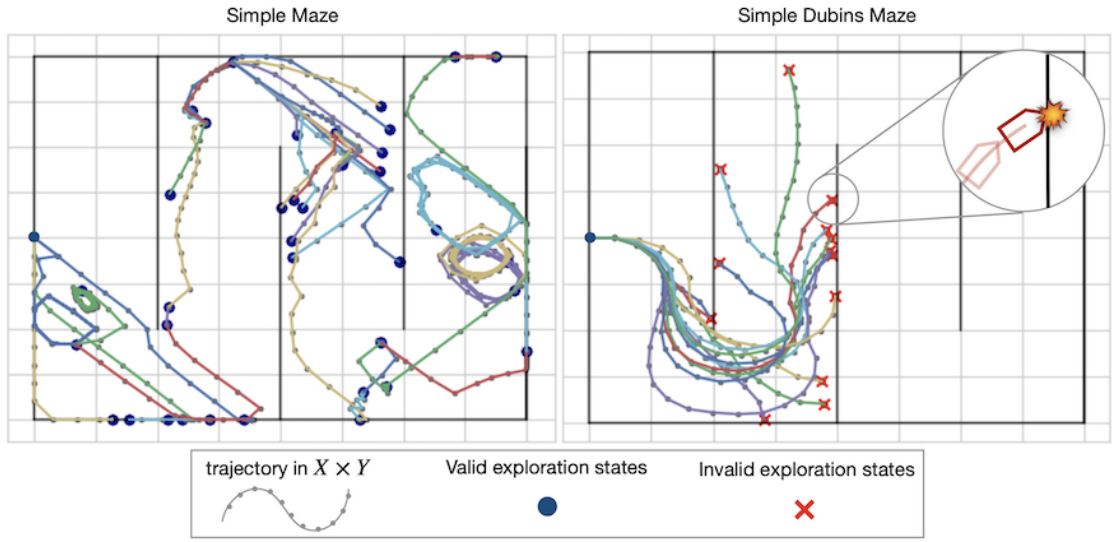


Figure 7.3: Visualisation of *invalid explorations states*. In the Simple Maze, each state is a *valid exploration state* as the agent can move in every direction. On the contrary, in the Dubins Maze, the agent is constrained to move forward. As a result, states colliding with the walls correspond to *invalid exploration states*.

results in no additional exploration as the agent is often blocked by the walls. These invalid exploration states are illustrated in Figure 7.3.

7.5 Conclusion

In this Section, we proposed NSSC, a skill-chaining-based variant of NS designed to tackle the non-locality of the expansion operator highlighted in Chapter 6. Although NSSC recovers a local expansion operator in a holonomic environment like the Simple Maze, limitations arise in non-holonomic environments. In particular, if we guide the exploration of the state space using a low-dimensional outcome space, exploration can be slowed down by useless expansions from *invalid exploration states* as expansions started from them result in no additional exploration.

We argue that *invalid exploration states* may block the exploration curriculum in any GCRL approach using a low-dimensional goal space (e.g. autotelic agents). In the next Chapters, we propose to bypass this limitation by learning from a demonstration.

8 Reinforcement Learning from a single demonstration with DCIL-I

8.1 Introduction

Given the recent successes of Reinforcement Learning (RL) (Sermanet et al., 2018; Akkaya et al., 2019) and Neuro-Evolution (NE) (Such et al., 2017; Colas et al., 2020) in robotics, one could be tempted to call upon these methods to circumvent the heavy burden of complex robotics behavior specification. A naive approach would consist of defining a success criterion and considering an objective function that would be positive when the task is achieved and null otherwise. Thus, one would let the learning algorithm discover the so-defined successful behavior on its own. Unfortunately, RL and NE algorithms struggle to solve precisely these Hard Exploration Problems where the target behavior is complex and the objective function corresponding to success is ill-defined (e.g. sparse or deceptive) (Matheron et al., 2019). For instance, an off-the-shelf RL algorithm cannot learn a control policy for verticalizing an underactuated humanoid robot if the reward signal is only received when the robot is standing.

As explained in Chapter 3.3.2, integrating intrinsic motivation in agents is a promising path toward the end-to-end resolution of such problems. However, current methods do not scale to the most complex continuous control tasks. In addition, they are often very sample-inefficient (Aubret et al., 2019).

In this context, leveraging expert demonstrations to bootstrap the learning process is a fruitful approach, as it efficiently guides and accelerates policy optimization (Ho and Ermon, 2016b; Kostrikov, Agrawal, Dwibedi, et al., 2018). In Imitation Learning (IL), the demonstrations are used to match the agent's behavior to the expert's one. However, many demonstrations containing both states and actions are generally required to perform IL (Pomerleau, 1991; Russell, 1998) successfully. Obtaining such demonstrations can be prohibitively difficult when dealing with complex systems in

real world environments. Moreover, actions are not always available. For instance, when demonstrations come from motion capture (Merel et al., 2018; X. B. Peng et al., 2018) or human guidance (Johns, 2021), only state-based trajectories are available. Only a few IL algorithms can work with a single demonstration (Resnick et al., 2018; Salimans and R. Chen, 2018; Dadashi et al., 2020). Even fewer, with a single state-based demonstration.

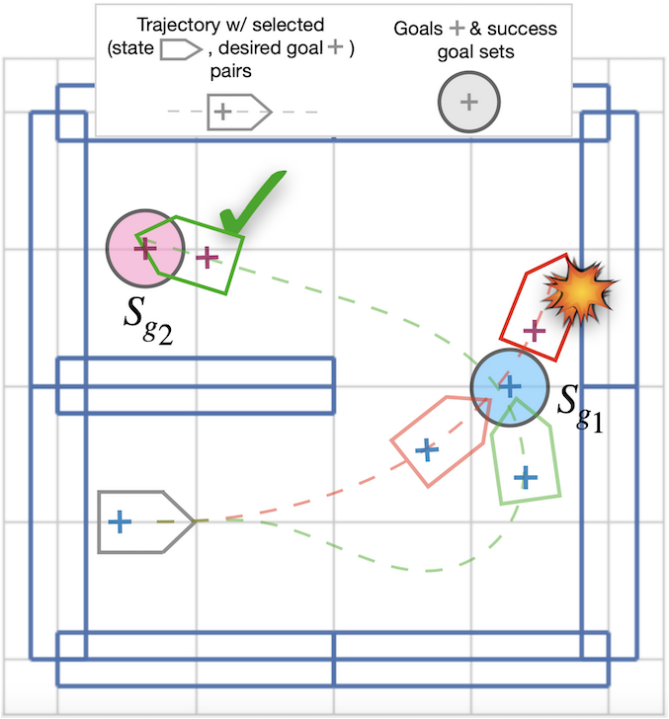


Figure 8.1: Illustration of the sequential goal-reaching problem when using low-dimensional projections of states as goals. In this toy 2D maze, the agent corresponds to a Dubins car (Dubins, 1957) with (x, y, θ) states. Goals g_i correspond to desired (x, y) positions but do not condition the orientation θ of the car. In the red trajectory, the car reaches goal g_1 with an orientation that is not compatible with the achievement of goal g_2. On the contrary, in the green trajectory, the trajectory of the car is adapted to reach the first goal with a valid orientation. The GCRL framework we present in this chapter is designed to learn policies that ensure targeting valid states when achieving a sequence of low-dimensional goals extracted from a demonstration.

In the following chapters, we focus on the problem of learning a control policy to solve complex robotics tasks with a very sparse reward (or in the absence of any reward) using a single demonstration. We tackle this problem by leveraging a sequential inductive bias in the way we model the task. First, we convert the demonstration into a succession of intermediate low-dimensional goals. Then, we use Goal-Conditioned Reinforcement Learning (GCRL) (see Section 3.4) to learn a policy to control the robot

to reach the intermediate goals sequentially. For instance, when learning a control policy to verticalize an underactuated humanoid robot, a sequence of goals may correspond to a succession of Cartesian positions of the torso of the humanoid from the ground to the desired height. By achieving these goals sequentially, the robot can stand up.

However, when dealing with a high-dimensional state space, reaching a low-dimensional goal does not fully condition the current state of the agent. This can raise issues if some intermediate goal reaching states are incompatible with reaching the next goal. For instance, consider a simple non-holonomic environment similar to the Dubins car (Dubins, 1957) navigating a maze in Chapter 6, as illustrated in Figure 8.1. In this environment, reaching a desired (x, y) position does not condition the arrival orientation θ of the car. This partial conditioning of the success state may be problematic when we want to reach successive (x, y) positions. Indeed, if the car reaches a goal close to a wall and is oriented towards it, it may not escape from hitting the wall before reaching the next goal.

Thus, when considering a sequence of goals corresponding to low-dimensional projections of high-dimensional states, it is essential to ensure that the agent achieves each goal by reaching states compatible with the achievement of the next goal, which we call *valid success states*. This is the key property ensured by the Divide & Conquer Imitation Learning I (DCIL-I) algorithm proposed in this chapter and the variants proposed in chapters 9 and 10.

In DCIL-I, the agent is encouraged to achieve the goals only via *valid success states* using two mechanisms. First, an overshoot mechanism is used to train the agent to reach a goal from non-demonstrated states. In addition, a chaining reward bonus propagates the value function of a goal into the value function of the previous goals of the sequence. We first evaluate our approach in a toy Dubins maze environment where the dynamics of the controlled system are constrained and show that our chaining mechanism plays a crucial role in ensuring the success of the method. Then, we compare DCIL-I with two competing algorithms that we present below, Backplay and PWIL, and we show that our approach is several orders of magnitude more sample efficient than the two baselines. We then turn to a more challenging Fetch environment where an object has to be grasped and put into a drawer with a simulated robotic arm, and demonstrate an even greater gain in sample efficiency compared to Backplay, the method used by Go-Explore on this benchmark.

8.2 Background

8.2.1 Valid success states

When an agent achieves a low-dimensional goal according to a distance-based reward, it may have reached various success states. This can be problematic if the agent needs to reach another goal but cannot reach it from certain success states, for instance, because of the non-holonomic constraints of the environment (see Figure 8.2). Thus, the set of success states $\mathcal{S}_g$ for a given goal g can be divided into valid (noted $\mathcal{S}_g^{valid}$) or invalid states for reaching the next goal.

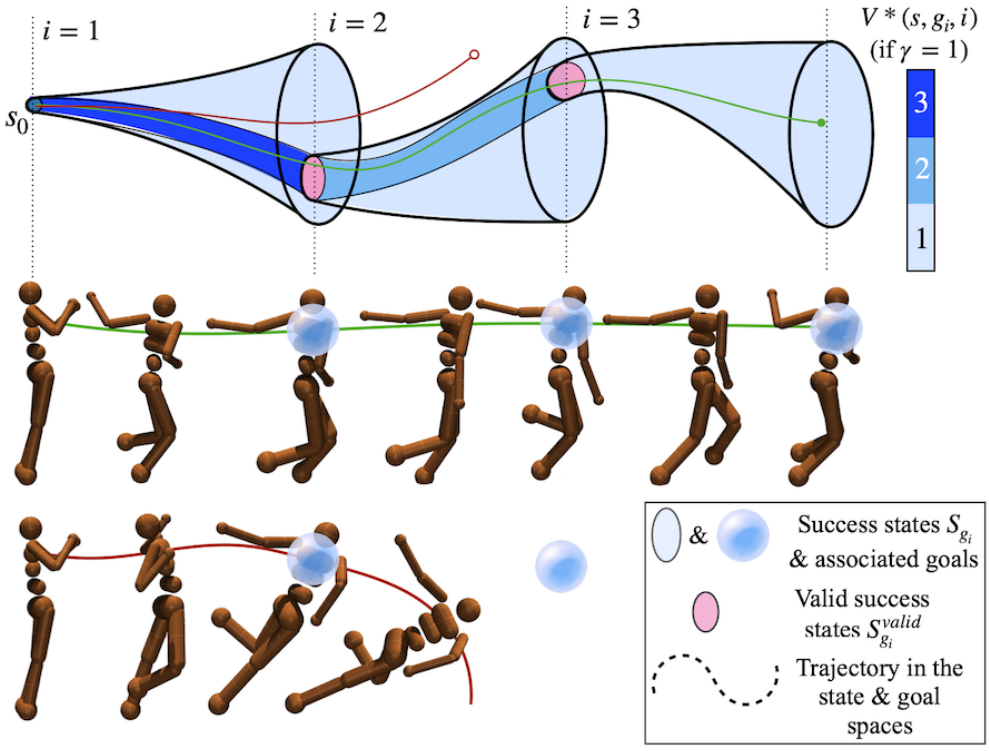


Figure 8.2: To successively reach each goal, the agent must transit between successive sets of valid success states. The non-terminal success states propagate the reward from each index i to the previous ones. **Top:** value propagation in a simplified 2D representation of the state space when $\gamma = 1$. In the shown environment, goals correspond to successive Cartesian positions of the torso of a humanoid. **Middle:** trajectory where the agent successfully transitions from one valid success state set to another. **Bottom:** trajectory where the agent jumps to achieve the first goal but does it by reaching an invalid success state which prevents it from reaching the next goal.

8.3 Related work

Our method performs imitation learning by leveraging a single demonstration, leverages a sequential goal-reaching bias, and focuses on reaching valid success states only using a value propagation mechanism. We cover the literature on these three topics taken separately.

8.3.1 IL from a single demonstration

IL is usually transformed into an optimization problem whose objective is to reproduce the behavior of an expert. It can be done directly in Behavioral Cloning (Pomerleau, 1991), which relies on regression to learn a policy that mimics the expert's actions. A more indirect approach is Inverse Reinforcement Learning (IRL) (Russell, 1998), which consists in estimating an unknown reward function from demonstrations of an expert considered optimal and training a policy using the learned reward function.

These approaches are severely limited by the necessity to measure the expert's actions and their typical need for many demonstrations. Specifically, with few demonstrations, BC tends to suffer from compounding error caused by distribution shift (Ross and D. Bagnell, 2010). In the case of IRL, it can be difficult to extract a reward from a unique demonstration. For instance, popular adversarial methods for IRL ((Ho and Ermon, 2016a; Henderson et al., 2018; Kostrikov, Agrawal, Levine, et al., 2018; Ding et al., 2019)) rely on a generator-discriminator architecture that may become unstable if the discriminator is not trained on a sufficient number of expert trajectories.

There are contexts in which demonstrations are rare or difficult to generate, but only a few deep learning-based methods can produce good results in this low-data regime. In our experimental validation, we mainly consider two of them: PWIL and Backplay.

PWIL

Primal Wasserstein Imitation Learning (PWIL) (Dadashi et al., 2020) is a recent IRL method that minimizes a greedy version of the Wasserstein distance between the agent and expert state-action distributions. The Wasserstein distance presents several good properties that have been proficiently used in the Deep Learning community (Arjovsky et al., 2017). PWIL solves an occupancy matching problem between an agent and the demonstration without relying on adversarial training, making it more stable than adversarial methods. Specifically, it defines an episodic reward function based

on the demonstration and performs IL by maximizing this reward without introducing an inner minimization problem as adversarial approaches do. PWIL achieves strong performances in complex simulated robotics tasks like humanoid locomotion using a single demonstration.

Backplay

The Backplay algorithm ((Resnick et al., 2018), (Salimans and R. Chen, 2018)), is an approach explicitly designed for IL from a single demonstration. It has been used in the robustification phase of the first version of the Go-Explore algorithm (Ecoffet et al., 2019b) to achieve state-of-the-art results on the challenging Atari benchmark Montezuma's revenge and in the Fetch problem that we tackle in Section 8.5.4. In Backplay, the goal is to reach the final state of the expert demonstration. The RL agent is initialized near the rewarding state and the starting state is gradually moved backward along the demonstration if it is successful enough in reaching the desired state. Backplay can be seen as a curriculum for RL approaches in the context of sparse reward and long-horizon control (Florensa et al., 2017).

8.3.2 Sequential Goal Reaching

The imitation objective we consider is not to precisely reproduce a trajectory, but to achieve the same goal(s) as the demonstration. We solve the single demonstration imitation problem by transforming a demonstration into a sequence of goal-reaching tasks. This amounts to a Divide & Conquer strategy that has similarities with the classical multiple shooting (MS) technique in Optimal Control (OC) (Bock and Plitt, 1984). In MS, an OC problem with boundary constraints is transformed into a sequence of simpler initial value problems over short time intervals connected by matching conditions to ensure the continuity of the solution.

Sequential goal achievement is also the central tool of several other RL methods. In Feudal Hierarchical Reinforcement Learning (HRL) methods (Dayan and Hinton, 1992; Nachum et al., 2018; A. Levy et al., 2019), a high-level policy constructs a sequence of goals on the fly by periodically assigning goals to a low-level goal-conditioned policy over the entire episode. However, jointly training a multi-stage policy induces training instabilities caused by non-stationary transition functions (Nachum et al., 2018; Hutsebaut-Buysse et al., 2022).

To achieve a difficult-to-reach goal, several approaches use the goal-conditioned value function of the agent to obtain a sequence of simpler goals (A. V. Nair et al.,

2018; Eysenbach et al., 2019; Nasiriany et al., 2019; Chane-Sane et al., 2021). However, these methods rely on an accurate value function which is rarely available in Hard-Exploration Problems. Their applicability either assumes that the agent can easily explore the entire state space or that the agent can be reset in uniformly sampled states. By contrast, we assume hard exploration environments with a sparse reward function and do not leverage a reset-anywhere property, as DCIL-I only resets along the demonstrated trajectory.

The policy-based version of the Go-Explore algorithm (Ecoffet et al., 2021) also guides the agent with a sequence of goals extracted from training trajectories. These goals are used to guide the agent back to previously visited states in order to continue the exploration process directly from advanced states. However, the authors did not dwell on the problem of invalid success states. In a recent extension of the algorithm (Gallouédec and Dellandréa, 2022), the authors bypass this problem by sub-sampling the sequence of goals.

In our method, we tackle this problem using value propagation, a mechanism also used in Option-based HRL, as described below.

8.3.3 Value propagation mechanisms

In Option-based HRL (Sutton et al., 1999; Precup, 2000) (see Section 7.1 for more details), when intra-option policies are conditioned on goals, HRL relies on sequential goal-reaching. Indeed, similarly to Feudal HRL, a sequence of goals is built on-the-fly by the policy over options. This view was adopted in Robust Deep Skill Chaining (R-DSC) (Bagaria, Senthil, Slivinski, et al., 2021) which trains goal-conditioned intra-option policies. In R-DSC, the value of an option is propagated to the value of the previous option in order to select the optimal goal for chaining these two successive options. Unlike DCIL-I, R-DSC only adapts the goal for better skill chaining and the intra-option policy is not adapted to reach valid success states only.

Some other option-based HRL methods (K. Y. Levy and Shimkin, 2011; Bacon et al., 2017) designed an MDP with an extended state space to facilitate switches between options by propagating the value from one to another. However, this framework has not yet been adapted to goal-conditioned policies. Moreover, these option-based HRL methods as well as R-DSC do not learn from demonstrations.

The similarities and differences between these different value propagation mechanisms is discussed in detail in the next chapter (see Section 9.3)

8.4 Methods

The Divide & Conquer Imitation Learning I (DCIL-I) algorithm is designed to solve a Goal-Guided Imitation (GGI) problem. In a GGI problem, instead of imitating the whole expert demonstration, we rely on the *Divide & Conquer* paradigm and divide the imitation problem into simpler separate goal-reaching tasks. This implies a small loss of generality as it relies on the assumption that demonstrations can be decomposed into a sequence of goal-conditioned tasks. Arguably, this assumption is often true in a robotic context, and as we show in Section 8.5, the GGI approach can significantly accelerate the IL process.

8.4.1 Goal-Guided Imitation

The Goal-Guided Imitation (GGI) framework can be formulated as a variant of GCRL. A set of N_{goals} goals $(g_i)_{i\in[0,N_{skills}]} \in \mathcal{G}^{N_{skills}}$ is extracted from a single expert demonstration $\tau_{demo} = \{s_0^{demo}, s_1^{demo}, ..., s_N^{demo}\}$ and the objective is to obtain a GCP that is able to reach each goal separately. This GCP is then used to sequentially reach each goal in order to complete the demonstrated behavior.

8.4.2 DCIL-I hypotheses

We formulate three main hypotheses in DCIL-I. We assume a weak form of *reset-anywhere*, that expert actions are not provided and that a definition of the goal space is given.

Reset

Training the GCP in DCIL-I (see Section 8.4.3) assumes that the agent can be reset in some selected states of the expert demonstration. A similar form of reset is assumed in Backplay which requires that the agent can be reset in each demonstrated state. Stronger reset assumptions like *reset-anywhere* where the agent can be reset in uniformly sampled states have also been considered in the GCRL literature (Nasiriany et al., 2019). PWIL is based on the more classical assumption of a unique reset, and BC does not require any reset at all.

No expert actions

Similarly to Backplay, DCIL-I learns solely from an expert trajectory in the state space. Learning from states only is crucial when the expert actions are difficult to collect (e.g. motion capture, human guidance). On the contrary, both PWIL and BC require state-action demonstrations.

Goal-space definition

As a GCRL-based method designed to be applied in high-dimensional state spaces, DCIL-I requires a definition of the goal space and the corresponding mapping from the state space to the goal space. No such assumption is made in Backplay, PWIL or BC as none of them solve a GCRL problem.

8.4.3 The Divide & Conquer Imitation Learning I algorithm

In DCIL-I, we extract a sequence of goals and initial states from the expert trajectory. The GCP is then trained to reach each goal using a DRL algorithm. Training for a goal boils down to starting in the associated initial state and completing a local rollout to reach the goal. While the agent train for a goal, it is encouraged to complete it by reaching states that are compatible with the execution of the achievement of the next one i.e *valid success states*. Finally, the agent can recover the expert behavior by reaching the sequence of goals sequentially. These different stages of DCIL-I are detailed in the four next sections and summarized in Algorithms 6 and 7.

Extracting goals from the expert trajectory

To extract the sequence of goals $\tau_{\mathcal{G}}$ from the demonstration, we project the demonstrated states in the goal space and split the demonstration into N_{goal} sub-trajectories of equal arc lengths ϵ_{dist}. For each sub-trajectory in the goal space, we extract its final elements and concatenate them to construct $\tau_{\mathcal{G}}$. We also extract the states associated with the initial elements of each sub-trajectories. Those states form a set of demonstrated states used by DCIL-I to reset the agent in a valid success state and to train it to reach the following goal (Hypothesis 8.4.2).

Training the GCP

To train the GCP to reach each goal in $\tau_{\mathcal{G}}$, DCIL-I runs a three-step loop.

Algorithm 6 DCIL-I - GCP training

1: **Input:** $\pi_\phi, Q, \bar{Q}, \tau_{demo}$ ▷ actor/critic/target critic networks & demonstration
2: $\{g_i\}_{i\in[1,N_{goals}]} \leftarrow$ extract_goals(τ_{demo})
3: $B \leftarrow []$ ▷ replay-buffer
4: $R \leftarrow \{i : [\,]\}_{i\in[1,N_{goals}]}$ ▷ successes/failures memory
5: **for** $n = 1 : N_{episode}$ **do**
6: $(s_0^i, g_i) \leftarrow select\_goal(R)$ ▷ **Step 1**
7: $s \leftarrow$ env.reset(s_0^i)
8: $t \leftarrow 0$
9: $success, done, timeout \leftarrow False, False, False$
10: **while** not done **do** ▷ **Step 2**
11: $a \sim \pi(a|s, g_i)$
12: $s' \leftarrow env.step(a)$
13: $r \leftarrow 0$
14: **if** $s' \in S_{g_i}$ **then** ▷ success
15: $success, done \leftarrow True, True$
16: $R[i] \leftarrow R[i] + [0]$
17: **else if** $t \geq T^n_{max}$ **then** ▷ failure
18: $success, timeout \leftarrow False, True$
19: $R[i] \leftarrow R[i] + [0]$
20: $B \leftarrow B + [(s, a, s', r, g_i, done, success, timeout)]$
21: $t, s \leftarrow t + 1, s'$
22: **if** success **then** ▷ **Step 3**
23: $success, done \leftarrow False, False$
24: $g_i \leftarrow$ next_goal(g_i) ▷ overshoot
25: $t \leftarrow 0$
26: SAC_update($\pi_\phi, Q, \bar{Q}, B$) ▷ Algo 7

Step 1 (line 6) DCIL-I selects a goal $g_i \in \tau_{\mathcal{G}}$ to train on (function select_goal in Algorithm 6) and resets the environment in the associated demonstrated state. Note that the selection of a goal is biased towards goals with a low ratio of success (line 6). We implemented such distribution using a fitness proportionate selection (Blickle and Thiele, 1996) where the fitness corresponds to the inverse of this ratio.

Step 2 (lines 10 to 21) DCIL conditions the GCP on g_i and the agent performs a rollout which is interrupted either if the agent reaches $\mathcal{S}_{g_i}$, if a time limit is reached or if the agent reaches a terminal state.

Algorithm 7 modified SAC update (+ HER + Chaining Reward Bonus)

Input: $\pi, Q, \bar{Q}, B$ ▷ actor/critic/target critic networks
$batch_{HER} \leftarrow HER(B)$ ▷ HER relabelling
$batch \leftarrow B$ ▷ no HER relabelling
for $(s, a, s', g_k, r, done, success, timeout)$ in $batch$ **do**
  if success **then** ▷ Chaining reward bonus
    $g_{k+1} \leftarrow$ next_goal(g_k)
    $r \leftarrow 1 + \bar{Q}(s', \pi(s', g_{k+1}), g_{k+1})$
$batch \leftarrow batch + batch_{HER}$
for $gradient\_step = 1 : N_{gradient\_step}$ **do** ▷ see (Haarnoja, Zhou, Abbeel, et al., 2018)
  $\pi, Q, \bar{Q} \leftarrow gradient\_step(\pi, Q, \bar{Q}, batch)$
Output: $\pi, Q, \bar{Q}$

Step 3 (lines 22 to 25) When the agent successfully reaches g_i, DCIL-I applies an overshoot mechanism (see Section 8.4.3) and returns to Step 2. Otherwise, the complete loop is repeated.

The rollouts are saved in a unique replay buffer. For each interaction with the environment, the saved transitions are sampled in a batch to perform a SAC update (Haarnoja, Zhou, Abbeel, et al., 2018) of the critic and actor networks including the chaining reward bonus (see Section 8.4.3). In a sampled batch of transitions, half of the transitions are relabelled using HER. This three-step loop is summarized in Algorithm 6.

Ensuring successful sequential goal reaching

To ensure successful sequential goal reaching after training for each goal independently, DCIL-I uses an *overshoot mechanism* and a *chaining reward bonus*.

Overshoot Successfully chaining goals requires that the agent achieves any goal by reaching states from which it is able to perform the next ones. When the agent achieves a goal, it can reach either *valid success states* in $\mathcal{S}_g^{valid}$ from which it can achieve the next goal or *invalid success states* from which it cannot (see Section 8.2.1). During a training rollout for a goal, if the agent reaches a success state, the overshoot mechanism immediately conditions the GCP on the next goal in $\tau_{\mathcal{G}}$ (function next_goal in Algorithm 6). The agent instantly starts a new rollout for this next goal from its current state.

With these overshoot rollouts, the agent can learn how to reach the next goal while starting from other states than the ones extracted from the demonstration. This is crucial for successful goal reaching as the agent is unlikely to achieve a goal by reaching the demonstrated success states exactly. However, in complex environments (e.g. non-holonomic environments), the pressure for fast goal achievement may force the agent to reach invalid success states. Therefore, another mechanism is necessary to encourage the agent to complete skills by only reaching valid success states.

Chaining reward bonus To encourage the agent to achieve goals by only reaching valid success states for the next goal, DCIL-I adds a chaining reward bonus to the sparse reward (3.23) received by the agent each time it successfully reaches a goal. The chaining reward bonus is defined as the goal-conditioned value function $V^{\pi}(., g_{i+1})$ of the agent, conditioned on the next goal in $\tau_{\mathcal{G}}$ (see Algorithm 7 for additional information on goal-conditioned value computation). Therefore, the modified reward function is defined as:

$$\bar{R}(s, a, s', g_i) = \begin{cases} 1 + V^{\pi}(s', g_{i+1}) \text{ if } s' \in \mathcal{S}_{g_i} \\ 0 \text{ otherwise.} \end{cases} \tag{8.1}$$

The idea behind this bonus is that valid success states should have a higher value than invalid ones. Indeed, by training the value function $V^{\pi}(., g_{i+1})$ with transitions extracted from successful episodes and successful overshoots from valid success states, the values $V^{\pi}(., g_{i+1}), V^{\pi}(., g_{i+2}), ...$ are propagated backward up to valid success states.

In complex environments, if $\mathcal{S}_{g_i}^{valid}$ only contains a few isolated valid success states (or even only the demonstrated state s_0^i), the chaining bonus reward may be difficult to propagate along the skills. A well-balanced entropy-regularized soft update of SAC, using either a hand-tuned entropy coefficient, forces the agent to explore diverse trajectories to reach $\mathcal{S}_{g_i}$ (Eysenbach and Levine, 2021). It helps the agent to eventually find a path towards a valid initial state and propagate the chaining bonus reward.

In Chapter 9, a more efficient value propagation mechanism is presented.

Retrieving the expert behavior

To reproduce the expert behavior, we reset the environment in the initial state of the expert demonstration. We condition the GCP on the first goal g_0 of $\tau_{\mathcal{G}}$. After completing this first skill, the agent is conditioned on g_1. This process is repeated for every skill K_i until the final goal is reached.

8.4.4 Value Clipping & discount factor choice for stable training

In an episodic setting, the length of a training rollout is limited. When the agent hits a time limit T_{max}, the rollout is truncated and the agent is reset. For maximum sample efficiency, one could be tempted to use a small time limit to perform short training rollouts only. However, in this case, the critic network may poorly generalize and impact the learning process of the goal-conditioned policy. Thus, one must take a few precautions while setting the time limit.

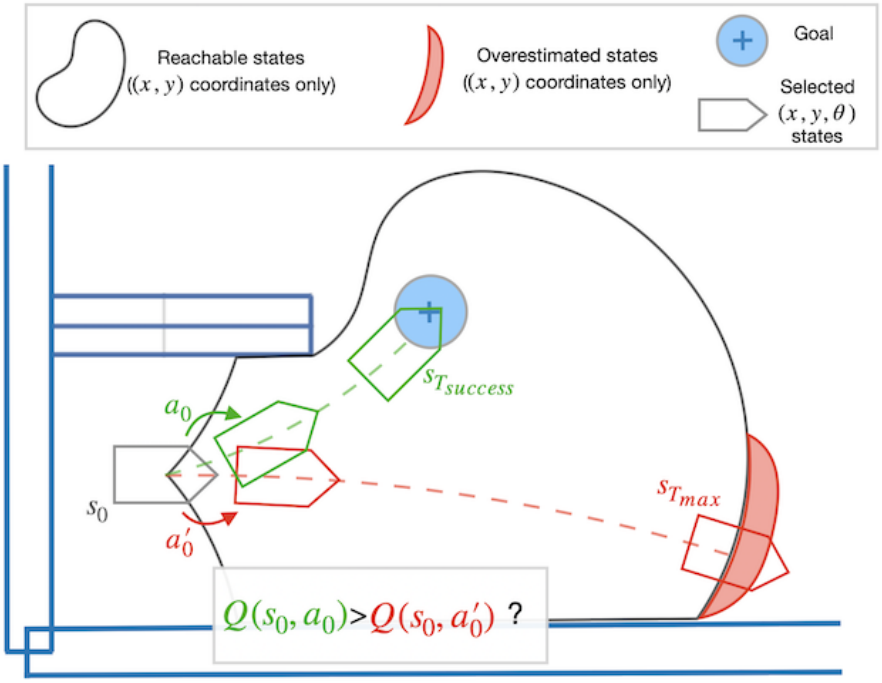


Figure 8.3: Illustration of the setup for time limit tuning in a maze similar to Figure 8.2. State $s_{T_{max}}$ is at the frontier of the reachable states (grey line) and belongs to a set of states with overestimated values (in red). The time limit T_{max} must be chosen carefully to ensure that the Q-value $Q(s_0, a_0)$ associated with an action a_0 directed towards the goal (in blue) is larger than another action a'_0 directed towards the overestimated states.

Any state $s_{T_{max}}$ that the agent reached while hitting the time limit is not a terminal state. Therefore, for a given transition $(s_{T_{max}-1}, a_{T_{max}-1}, s_{T_{max}}, r_{T_{max}-1})$ to $s_{T_{max}}$, the critic network is updated to match the full TD target $V_{target}(s_{T_{max}-1}) = r_{T_{max}-1} + \gamma V(s_{T_{max}})$.

However, if $s_{T_{max}}$ belongs to the frontier of the reachable set of states as in Figure 8.3, its value is never updated and depends on the local generalization of the critic network. As it is difficult to anticipate how the output of a neural network evolves outside the training distribution, poor estimates of the value $V(s_{T_{max}})$ can appear. For instance,

$V(s_{T_{max}})$ may be overestimated. In that case, the overestimated value may eventually distract the agent from the goal. To limit the impact of such poor generalization, one may use a value clipping mechanism and adjust T_{max} accordingly.

Value clipping Knowing the number of goals in $\tau_{\mathcal{G}}$ and assuming a discount factor $\gamma = 1$, we know that $R_{max} = |\tau_{\mathcal{G}}|$ is the maximum cumulative reward that the agent can collect. Therefore, it is an upper bound for the value of any state.

In DCIL-I, during any critic update, the value of the next state is clipped using this upper bound in order to cap the value of potentially overestimated states at the frontier of the subset of reachable states. Therefore, the TD target for a transition to the frontier of the set of reachable states is defined as

$$V_{target}(s_{T_{max}-1}) = r_{T_{max}-1} + \gamma \min(V(s_{T_{max}}), R_{max}). \tag{8.2}$$

Tuning the time limit With the upper bound of the value in mind, we can tune the time limit to ensure that, in the presence of clipped value overestimation, optimal actions remain those that guide the action to the goal. This boils down to respecting the following inequality from which we can derive a condition for the minimal length of a training rollout. The intuition behind this inequality is illustrated by Figure 8.3 in a toy maze.

$$\begin{aligned} & Q(s_0, a_0) > Q(s_0, a_0') \\ \implies & \gamma^{T_{success}} > \gamma^{T_{max}} V(s_{T_{max}}) \\ \implies & \gamma^{T_{success}} > \gamma^{T_{max}} R_{max} \\ \implies & T_{max} > -\frac{\log(R_{max})}{\log(\gamma)} + T_{success}. \end{aligned} \tag{8.3}$$

For instance, for a successful trajectory length of $T_{success} = 15$ steps, a maximum clipped value of $R_{max} = |\tau_{\mathcal{G}}| = 5$ (corresponding to a sequence of 5 goals) and a discount factor $\gamma = 0.90$, the time limit should be at least $T_{max} = 30$ steps.

Careful adjustment of the time limit is essential to run DCIL-I. Indeed, in practice, if T_{max} is too small, the agent may be attracted toward overestimated states and catastrophically forget how to reach certain goals, which prevents robust sequential goal-reaching.

8.5 Experiments

In this section, we introduce the experimental setup used to evaluate DCIL-I (Section 8.5.1), we present an ablation study of the two main components of DCIL-I (Section 8.5.2) and we compare DCIL-I to three baselines: BC, Backplay and PWIL (Section 8.5.3). The code is based on Stable Baselines3 (Raffin et al., 2021) and provided here: https://github.com/AlexandreChenu/dcil.

8.5.1 Experimental setup

We evaluate DCIL in two environments: the *Dubins Maze* environment that we introduce and the *Fetch* environment presented in (Ecoffet et al., 2021).

Dubins Maze

In the Dubins Maze environment, the agent controls a Dubins car (Dubins, 1957) in a 2D maze. The state $s = (x, y, \theta) \in X \times Y \times \Theta$ where (x, y) are the coordinates of the center of the Dubins car in the 2D maze and θ is its orientation. The forward speed of the vehicle is constant with value 0.5 and the agent only controls the variation $\dot{\theta}$ of the orientation of the car. The goal space associated with this environment is $X \times Y$. In the absence of a desired orientation in the goal conditioning the GCP, the agent can easily reach a success state for a goal with an orientation invalid for the next goal in $\tau_{\mathcal{G}}$. Demonstrations for this environment are obtained using the Rapidly-Exploring Random Trees algorithm (LaValle et al., 1998).

Fetch

We also evaluate DCIL-I in the simulated grasping task for an 8-DoF deterministic robot manipulator. This environment was presented in the First Return then Explore paper (Ecoffet et al., 2021). The objective is to grasp an object initialized in a fixed position on a table and put it on a shelf. The state is a 604-dimensional vector and contains the configuration and the velocities of each element in the environment (robot, object, shelf, doors...) as well as the contact Boolean evaluated for each pair of elements. On the opposite, the goal only corresponds to the concatenation of the 3D coordinates of the end-effector of the manipulator with a Boolean indicating whether the object is grasped or not. Therefore, the agent may complete a skill prior to contact with the object with an invalid state (e.g. with an orientation or a velocity that prevents grasping). Demonstrations are collected using the exploration phase of the Go-Explore algorithm (Ecoffet et al., 2019b).

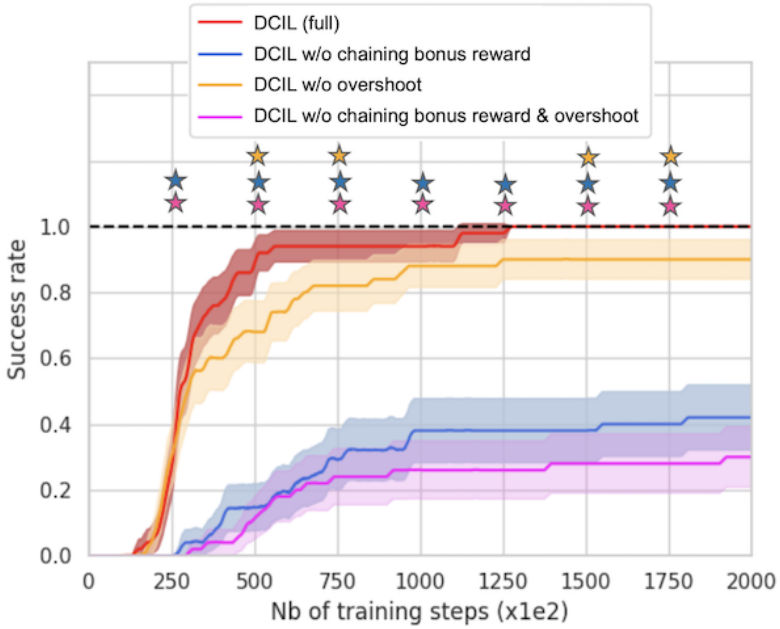


Figure 8.4: Ablation study of DCIL in the Dubins Maze environment. We evaluate the success rate of four versions of DCIL-I (full, w/o overshoot, w/o chaining bonus reward, w/o both) throughout training. Means and standard deviations range over 30 total runs (3 random seeds for 10 expert trajectories). Stars indicate significant differences over DCIL-I (full) as reported by Welch's t-test with $\alpha = 0.05$ (Colas et al., 2019).

Baselines

We compare DCIL-I to three IL methods. The first baseline is a naive BC method using a single demonstration. The two others are state-of-the-art methods that are proficient in the context of imitation from a single demonstration: Backplay ((Resnick et al., 2018; Salimans and R. Chen, 2018)) and PWIL (Dadashi et al., 2020). A comparison of the different key assumptions required by each algorithm is detailed in Section 10.3.4. For PWIL, we used the implementation provided by the authors with the same hyperparameter. For Backplay, we re-implemented it to evaluate it in the Dubins Maze and extracted the results obtained for the Fetch environment in (Ecoffet et al., 2021).

8.5.2 Ablation study

Using the Dubins Maze environment, we compare the full version of DCIL-I to variants without the chaining reward bonus, without the overshoot mechanism and without both.

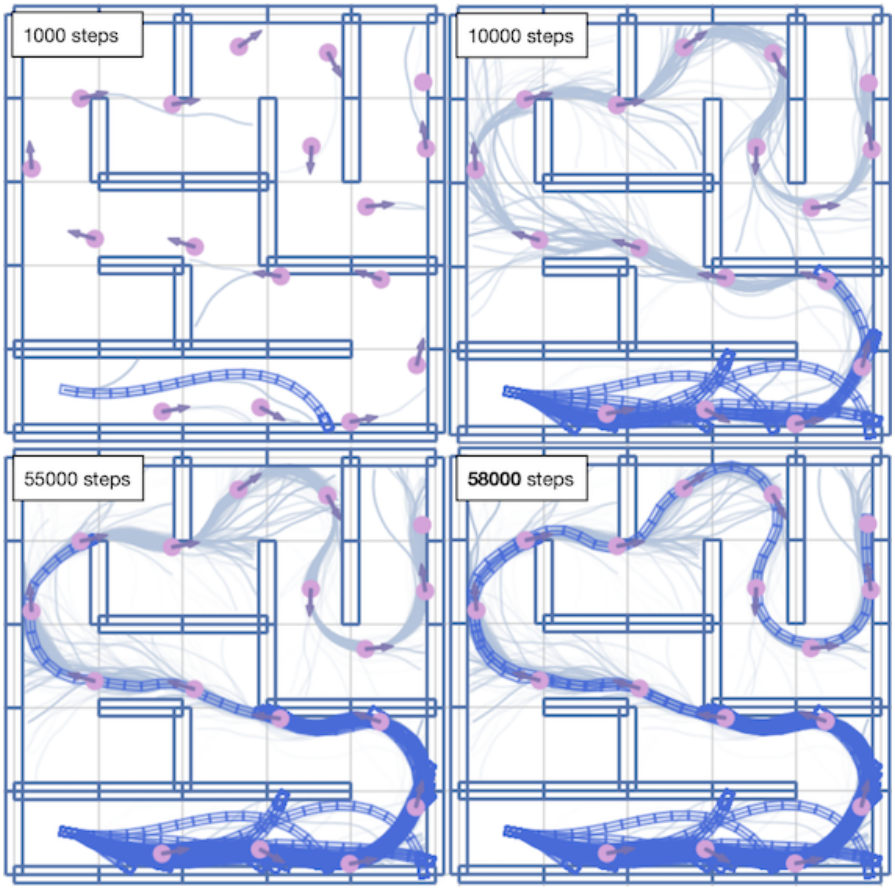


Figure 8.5: Training run of DCIL-I in the Dubins Maze. $\tau_{\mathcal{G}}$ is represented by pink disks and the associated initial states are represented by purple arrows. Grey lines correspond to skills training rollouts. Blue car trajectories correspond to the agent skill chaining every 1000 training steps.

The performance shown in Figure 8.4 is evaluated using the proportion of runs that solved the maze depending on the number of training steps. First, we can notice that the chaining reward bonus is critical to chain the skills and achieve a high success rate. Indeed, the two variants of DCIL-I using the chaining reward bonus (DCIL-I full and DCIL-I w/o overshoot) outperform the other two variants. Besides, only the full version of DCIL-I recovers the expert behavior in 100% of trials. Finally, DCIL-I also benefits from the overshoot mechanism as its success rate increases faster than DCIL-I w/o overshoot during the 50.000 first training steps.

Figure 8.5 presents a training run of the full version of DCIL-I. Although the GCP is training for each goal separately, it is able to chain them in order to recover the expert behavior and navigate the maze. In order to visualize how the chaining reward bonus encourages the agent to achieve goals by reaching valid success states, we evaluated

DCIL-I and DCIL-I w/o chaining bonus reward in a simplified version of the Dubins Maze where the agent only has two goals to reach sequentially. As Figure 8.6 shows, the chaining reward bonus increases the value of the states with a similar orientation to the states in the expert demonstration and results in successful chaining of the two skills.

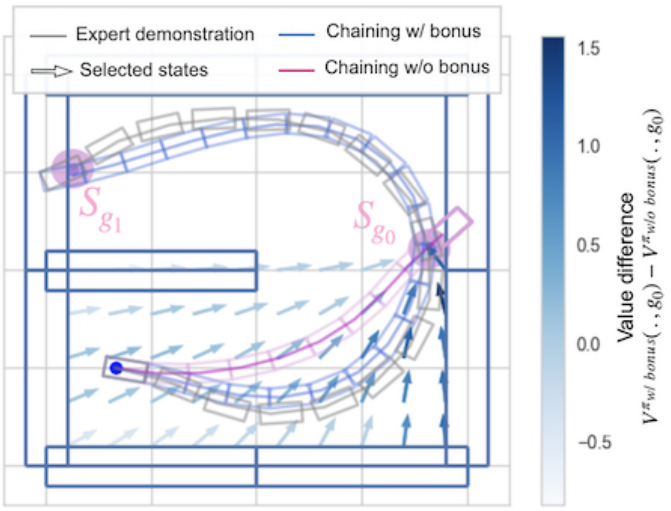


Figure 8.6: Learning how to achieve the first goal (reaching $\mathcal{S}_{g_0}$) without chaining reward bonus prevents the agent from achieving it by reaching a valid initial state for the second skill (reaching $\mathcal{S}_{g_1}$) as illustrated by the purple trajectory. The chaining reward bonus increases the value of the states with an orientation similar to the states of the expert demonstration (in grey) as shown by the difference between the g_0-conditioned values learned with a chaining reward bonus ($V^{\pi_{w/bonus}}(.,g_0)$) and without ($V^{\pi_{w/o\ bonus}}(.,g_0)$).

8.5.3 Comparison to baselines in Dubins Maze

Figure 8.7 evaluates how DCIL-I performed when trained on $1e6$ training interactions with the Dubins Maze compared to the three selected baselines. As the three baselines do not solve the maze after $1e6$ training interactions, we measure the progression through the maze. The maze is decomposed into 23 zones. The agent starts in zone 0 and the end of the maze is zone 22. DCIL-I is the only method to solve the maze within the allocated budget. It requires at most 10^5 training interactions. This is mainly due to the fact that DCIL-I trains on very short rollouts compared to PWIL which trains on fixed-length episodes and Backplay which trains on increasing-length episodes.

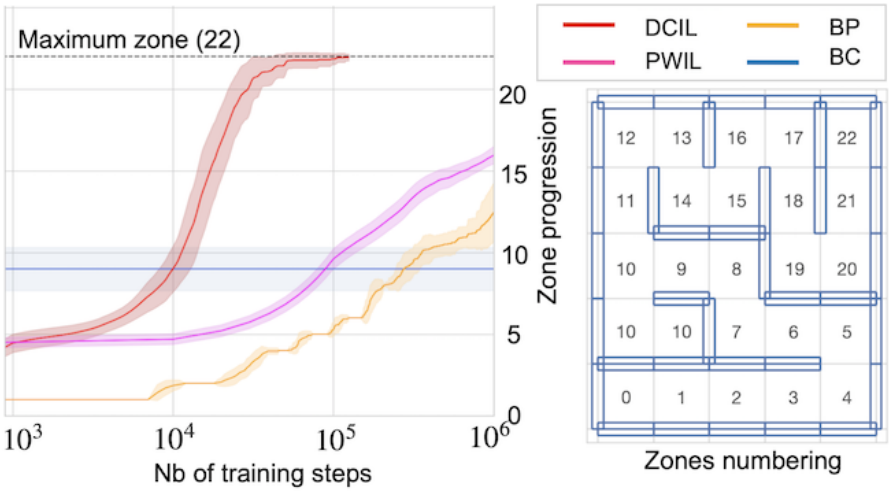


Figure 8.7: Comparison of DCIL-I to Backplay, PWIL and BC with a single demonstration. We evaluate the progression through the Dubins Maze Environment (left) using 22 zones (right). For DCIL-I, PWIL and BC, the progression corresponds to the maximum zone reached during evaluation. For Backplay, the progression corresponds to the difference between the highest zone number and the zone from which the agent started. Means and standard deviations range over 3 random seeds for 10 different expert trajectories (30 total runs for each variant) for each method.

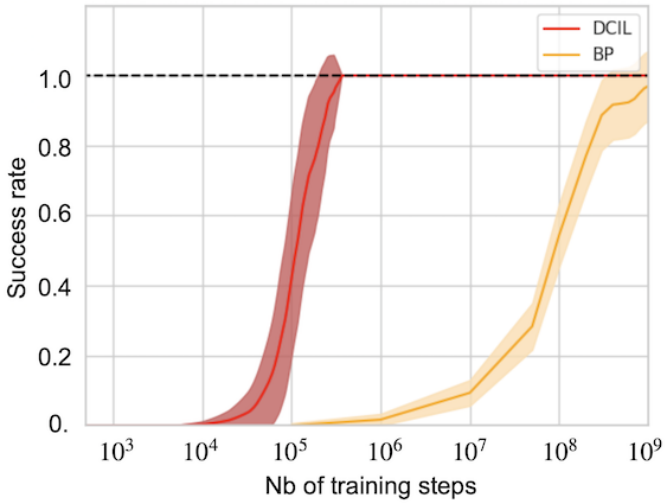


Figure 8.8: Comparison of DCIL-I to Backplay in the Fetch environment. The results of DCIL-I present the mean and standard deviation over 5 seeds for 5 expert demonstrations (25 total runs). The results of Backplay are extracted from (Ecoffet et al., 2021) and could not be reproduced despite running the author's code with the same hyper-parameters. For DCIL-I, in ~ 10% of the runs, the critic networks used in SAC diverge which results in failed runs. Diverging runs are not presented here.

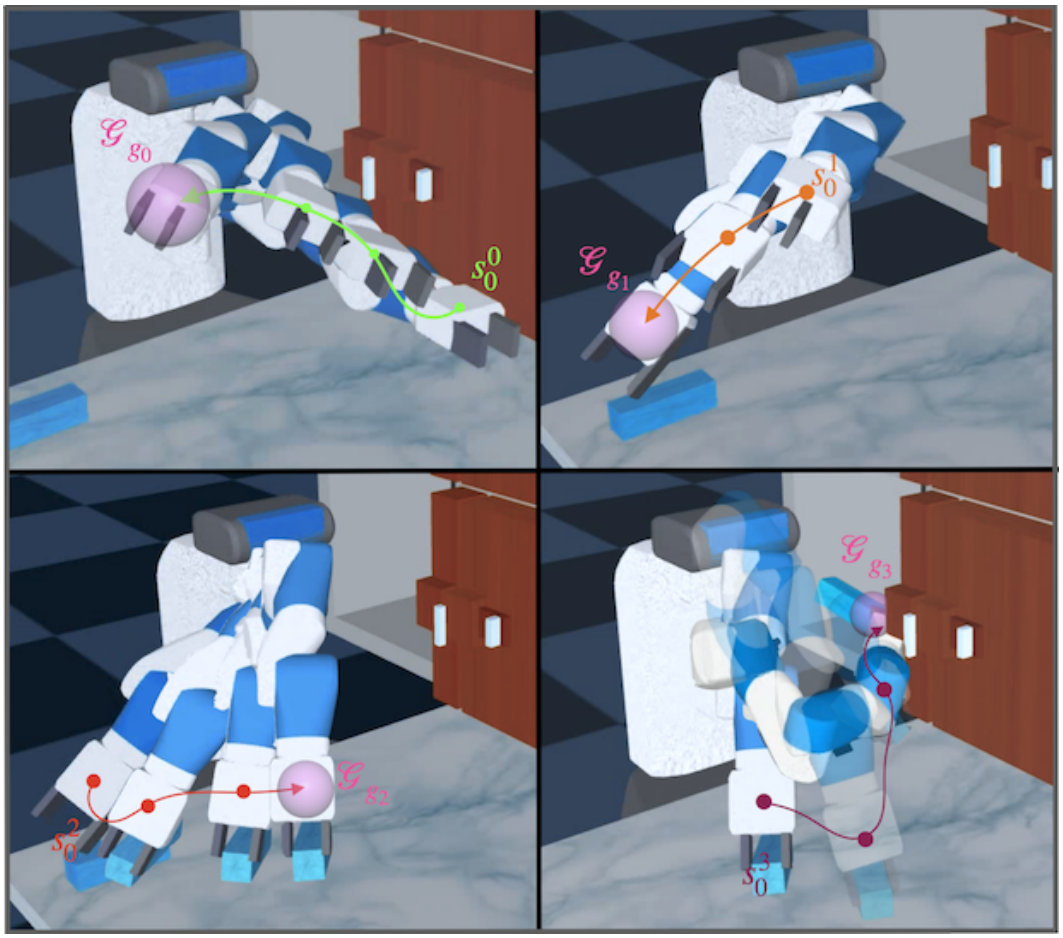


Figure 8.9: Sequential reaching of four goals learned with DCIL I to solve a simulated transportation task. The pink sphere represents the success zones defined in the goal space. The goal space contains the Cartesian positions of the end effector and a Boolean indicating if the object is grasped.

8.5.4 Scaling to a complex object manipulation task

While the experiments in the Dubins Maze demonstrate the performance of DCIL-I in a low-dimensional environment, we finally test our approach in the Fetch environment where observations are 604-dimension vectors and the transition function involves much more complicated dynamics.

Figure 8.8 presents the performance of DCIL-I compared to Backplay (Ecoffet et al., 2021). As in the Dubins Maze, DCIL-I solves the Fetch task three orders of magnitude faster than Backplay. DCIL-I learns the full behavior by training only on short rollouts. The grasping behavior resulting from the sequence goal reaching is illustrated in Figure 8.9.

Discussion & Conclusion

In this chapter, we have introduced Divide & Conquer Imitation Learning I (DCIL-I), an imitation learning algorithm solving long-horizon tasks using a single demonstration. DCIL-I relies on a sequential inductive bias and adopts a divide & conquer strategy to learn successive goals that, chained together, solve the long-horizon task. In order for a goal-conditioned policy to learn how to reach each goal individually and, then, sequentially reach them to recover the expert behavior, we introduced an *overshoot mechanism* and a *chaining reward bonus* that encourage the agent to prepare for the rest of the sequence while reaching a goal. We highlighted the key contribution of both mechanisms in the performance of DCIL-I by conducting an ablation study in a maze environment with a Dubins car. Moreover, we showed the efficiency of DCIL-I by comparing it to three IL baselines and by successfully applying it to a complex manipulation task. Compared to the baselines, we obtain an improvement of sample efficiency of several orders of magnitudes which will be critical when applying the method to physical robots.

In the next chapter, we introduce an original GCRL framework where a state is augmented with the index of the goal currently targeted. Using this new framework, we propose DCIL-II, a variant of DCIL-I that better propagates the value in the sequence of goals and achieves higher sample efficiency.

9 Improving the value propagation mechanism with DCIL-II

9.1 Introduction

In this chapter, building on work done with the DCIL-I algorithm in the previous chapter, we present a Markov Decision Process (MDP) with an extended state space that includes the sequence of goals extracted from the demonstration. This MDP is designed to encourage the agent to achieve the goals only via valid success states. Our approach ensures that the reward received by the agent for reaching a goal is propagated to the valid success states of the previous goals in order to make sure that their value is higher than the value of invalid ones. To improve the exploration of the state space while training, we transform our MDP into a goal-conditioned MDP (GC-MDP) and adapt the relabelling mechanism of Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) to this context. HER relabels the transitions of an episode by replacing the goal initially intended by the agent by the goal it accidentally achieved.

Based on this GC-MDP framework, the DCIL-II algorithm is able to efficiently learn a goal-conditioned policy that achieves the full sequence of intermediate goals until task completion. The value propagation mechanism adopted in DCIL-II is much more efficient than the bonus of reward of DCIL-I. As a result, DCIL-II outperforms DCIL-I in a large set of robotics tasks ranging from the grasping task presented in Chapter 8 to locomotion tasks with under-actuated humanoid robots.

The chapter is organized as follows. We first present the GCRL framework and the method in Section 8.4. In Section 9.3, we discuss the similarities between DCIL-II and other methods relying on a value propagation mechanism, including DCIL-I. In Section 9.4, we evaluate DCIL-II in several challenging robotic simulation scenarios and compare it to state-of-the-art algorithms able to learn complex behaviors by

leveraging a single demonstration.

9.2 Methods

In this section, we present a general framework where, given a sequence of low-dimensional goals extracted from a single demonstration, as in DCIL-I, an agent must learn to reach all goals sequentially to perform a complex task. We then derive the Divide & Conquer Imitation Learning II (DCIL-II) algorithm which builds on the framework to obtain such an agent.

9.2.1 Problem statement

DCIL-II is designed to solve a GCRL problem. However, in order to efficiently deal with the sequence of goals extracted from the demonstration, we use a slightly different framework from the one presented in Chapter 8.

An MDP with implicit sequential goals

Given an underlying MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, p, \gamma)$ with an uninformative reward function R that is non-zero only when the agent reaches a hard-to-reach desired goal, we consider a sequence of low-dimensional goals $\tau_{\mathcal{G}} = \{g_i\}_{i \in [0, N_{goals}]}$ leading to the desired goal.

We extend the state space to include the index of the current goal that the agent must reach. Thus, at step t in a trajectory, an extended state is described by a (s_t, i_t) pair where i_t is the index of the current goal. We adapt the reward function, the transition probability distribution and the discount function as follows.

The index-dependent reward function is defined as

$$R_{seq}(\cancel{s_t}, \cancel{a_t}, s_{t+1}, i_t) = \begin{cases} 1 \text{ if } s_{t+1} \in \mathcal{S}_{g_{i_t}}, \\ 0 \text{ otherwise.} \end{cases} \tag{9.1}$$

As shown with the barred s_t and a_t, this function only depends on the next underlying state and the index of the current goal. It rewards the agent when it reaches the success states associated with its current goal.

The agent must reach the successive goals sequentially. There might be different approaches to deal with this sequentiality constraint, but in the implementation

presented here, the index is automatically incremented after a successful transition:

$$i_{t+1} = \begin{cases} i_t + 1 \text{ if } s_{t+1} \in \mathcal{S}_{g_{i_t}}, \\ i_t \text{ otherwise.} \end{cases} \tag{9.2}$$

These automatic switches of indices are included in the joint transition probability which is defined as

$$\begin{aligned} & p_{seq}(s_{t+1}, i_{t+1} | s_t, a_t, i_t) \\ & \quad = p(i_{t+1} | s_{t+1}, \cancel{s_t}, \cancel{a_t}, i_t) p(s_{t+1} | s_t, a_t, \cancel{i_t}) \\ & \quad = p(i_{t+1} | s_{t+1}, i_t) p(s_{t+1} | s_t, a_t), \end{aligned} \tag{9.3}$$

with $p(s_{t+1}|s_t, a_t)$, the index-independent state transition probability of the underlying MDP and $p(i_{t+1}|s_{t+1}, i_t)$ the index transition probability. This latter corresponds to a deterministic distribution as the index at step $t+1$ is a constant random variable when conditioned on the next state and the current index. It can be defined as follows:

$$\begin{aligned} p(i_t + 1 | s_{t+1}, i_t) &= R_{seq}(s_{t+1}, i_t), \\ p(i_t | s_{t+1}, i_t) &= 1 - R_{seq}(s_{t+1}, i_t). \end{aligned} \tag{9.4}$$

When the agent reaches the final goal of the sequence, it enters a terminal state. So we define the discount function as

$$\gamma_{seq}(s) = \begin{cases} 0 \text{ if } s \in \mathcal{S}_{g_{N_{goal}}}, \\ \gamma \text{ otherwise.} \end{cases} \tag{9.5}$$

This function makes the success states associated with the intermediate goals in $\tau_{\mathcal{G}}$ non-terminal except for the final goal.

As discussed in Chapter 8, reaching a goal corresponding to a low dimensional projection of a state does not fully condition the state of the agent. In particular, if some success states are invalid, the agent must avoid them.

The definition of the discount function in (9.5) implicitly encourages the agent to

reach each goal by only reaching valid success states. Indeed, non-terminal success states associated with intermediate goals in $\tau_{\mathcal{G}}$ are used to propagate the value of the next goal to the previous one via the value approximation $\tilde{V}$ of successful transitions. Using (9.1) and (9.5) we get

$$\begin{aligned}\tilde{V}(s_t, i_t) &= R_{seq}(s_t, a_t, s_{t+1}, i_t) + \gamma_{seq}(s_{t+1}) V^{\pi}(s_{t+1}, i_t + 1) \\ &= 1 + \gamma V^{\pi}(s_{t+1}, i_t + 1).\end{aligned} \tag{9.6}$$

After enough training, the value $V^{\pi}(., i_t + 1)$ of valid success states associated with the next index should be larger than the value of invalid ones as the agent was only rewarded for trajectories starting from the former and not from the latter.

Therefore, both valid and invalid success states receive the sparse reward for reaching the current goal, but valid ones benefit from the propagation of a larger value from the next state. This difference is illustrated in Figure 8.2.

Using the extended state space, adapted reward function, transition probability and discount function defined above, we end up with an alternative MDP $\mathcal{M}_{seq} = (\mathcal{S}_{seq}, \mathcal{A}, R_{seq}, p_{seq}, \gamma_{seq})$. In this MDP, we condition the policy implicitly on the goals of the sequence by using their indices in the extended state space.

The objective in $\mathcal{M}_{seq}$ is to obtain a policy that maximizes the expected cumulative rewards:

$$\mathop{\mathbb{E}}_{\pi}[\sum_{t} R_{seq}(s_{t+1}, i_t) \prod_{k=0}^{t} \gamma_{seq}(s_k)]. \tag{9.7}$$

Maximizing this equation boils down to reaching each goal in $\tau_{\mathcal{G}}$ by passing through each set of valid success states.

Nevertheless, learning how to reach each goal using a sparse reward like R_{seq} can be challenging. Moving to GCRL, we can address this issue by leveraging the relabeling mechanism of HER.

Moving to GCRL to improve exploration

In Section 9.2.1, we defined a traditional MDP with an extended state. We now call upon the GCRL framework. In addition to implicit goal-conditioning via the goal

indices, we now also explicitly condition the policy $\pi(a|s,i,g_i)$ on the current goal g_i. As a result, we can use a relabelling mechanism like HER to reward transitions towards unintended goals and improve the exploration of the agent while it is training.

As with HER, during policy updates, a failed transition can be artificially relabeled as successful and the missed goal can be replaced by an unintended goal achieved later in the trajectory. However, unlike traditional GCRL, the goal changes along the trajectory as the agent must reach each goal sequentially. To include the automatic goal switches defined in (9.2) and (9.4) and let the value propagate between successive goals as in (9.6), we must relabel the next goal and next index too. Thus, in a failed transition relabeled as successful, the original next goal is replaced by the next goal in the sequence according to the current index. Besides, the next index corresponds to the incremented current index (see Figure 9.1).

To deal appropriately with relabelling in this context, we distinguish four different types of transitions $(s_t,i_t,g_t) \rightarrow_{a_t} (s_{t+1},i_{t+1},g_{t+1})$: original failed transitions 1, original successful transitions 2, transitions relabeled as failed for artificial goal $\bar{g}_t$ 3 or transitions relabeled as successful for artificial goal $\bar{g}_t$ 4. For each type of transition,

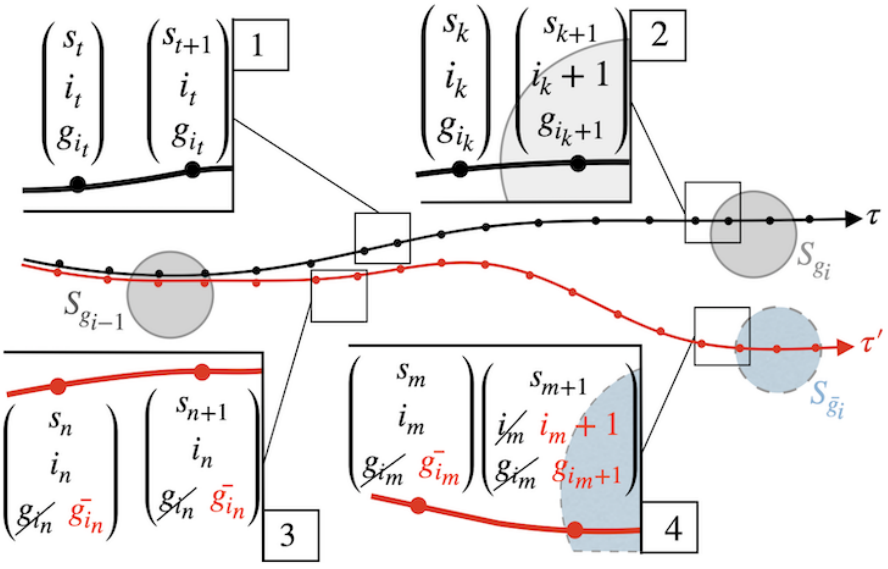


Figure 9.1: Automatic switches of goals and indices. When the agent reaches its current goal g_i as in the black trajectory, the current index i is incremented and the goal is switched to the next one in the sequence 2. When the agent fails to reach the current goal or relabeled goal $\bar{g}_i$, the goal and the index keep the same 1 & 3. For transitions relabeled as successful as in the red trajectory with a relabeled goal $\bar{g}_i$, the current and next indices and goals are relabeled to include these automatic switches 4.

the current and next goals and indices vary according to

$$(g_{t+1}, i_{t+1}) = f_{\text{next goal \& index}}(s_{t+1}, g_t, i_t) \tag{9.8}$$

with the $f_{\text{next goal \& index}} : S \times G \times \mathbb{N} \rightarrow G \times \mathbb{N}$ function returning the next goal and index given the current ones and the next state. This function switches to the next goal in the sequence according to the current index when the current goal is reached:

$$f_{\text{next goal \& index}}(s_{t+1}, g_t, i_t) := \begin{cases} (g_{i_t+1}, i_t + 1) \text{ if } s_{t+1} \in \mathcal{S}_{g_{i_t}} \\ (g_{i_t}, i_t) \text{ if } s_{t+1} \notin \mathcal{S}_{g_{i_t}}. \end{cases} \tag{9.9}$$

Using $f_{\text{next goal \& index}}$ to replace the next goal and index in both original and relabeled transitions, one can leverage HER in sparse reward contexts.

In addition, we also get value approximations similar to (9.6) for original successful transitions $\boxed{2}$:

$$\tilde{V}(s_t, i_t, g_{i_t}) = 1 + \gamma V^{\pi}(s_{t+1}, i_t + 1, g_{i_t+1}) \tag{9.10}$$

and transitions relabeled as successful for artificial goal $\bar{g}$ $\boxed{4}$:

$$\tilde{V}(s_t, i_t, \bar{g}) = 1 + \gamma V^{\pi}(s_{t+1}, i_t + 1, g_{i_t+1}) \tag{9.11}$$

Thus, both types of transitions propagate the value associated with the next goal $V(., i_t + 1, .)$ in $\tau_{\mathcal{G}}$ into the value associated with the current one $V(., i_t, .)$. Therefore, whatever the goal g conditioning the policy $\pi(a|s, i, g)$, the agent is always encouraged to prepare for the next goals in $\tau_{\mathcal{G}}$ while reaching the current one.

Note that if we decided to remove the index in the state to recover the usual GCRL state space (see Chapter 3.4) and to condition the policy $\pi(a|s, g)$ on the explicit goal only, we could not use a similar reward propagation mechanism.

Indeed, let's consider two failed training trajectories where the agent aimed for two different goals (g_i, g_j) in $\tau_{\mathcal{G}}$ and transited by the same states (s_t, s_{t+1}). In a transition relabeled as successful for artificial goal $\bar{g} = p_{\mathcal{G}}(s_{t+1})$, the relabeled next goals (g_{i+1}, g_{j+1}), defined according to the position of original goals in $\tau_{\mathcal{G}}$, are different. However, without any index, the states and the relabeled current goals $(\bar{g}_i, \bar{g}_j) = (\bar{g}, \bar{g})$

would be the same.

In this context, the value approximation $\tilde{V}$ of the same state-goal pair $(s_t, \bar{g})$ would be approximated differently depending on the chosen transition as the value of the next state would be conditioned differently.

For the transition originally aiming for g_i, we would have

$$\begin{aligned} \tilde{V}(s_t, \bar{g}) &= R_{seq}(s_t, a_t, s_{t+1}, \bar{g}) + \gamma_{seq}(s_{t+1}) V^{\pi}(s_{t+1}, g_{i+1}), \\ &= 1 + \gamma V^{\pi}(s_{t+1}, \underline{g_{i+1}}), \end{aligned} \tag{9.12}$$

whereas for the transition originally aiming for g_j, we would have

$$\begin{aligned} \tilde{V}(s_t, \bar{g}) &= R_{seq}(s_t, a_t, s_{t+1}, \bar{g}) + \gamma_{seq}(s_{t+1}) V^{\pi}(s_{t+1}, g_{j+1}), \\ &= 1 + \gamma V^{\pi}(s_{t+1}, \underline{g_{j+1}}). \end{aligned} \tag{9.13}$$

Therefore, using our relabelling mechanism for reward propagation between goals in a GC-MDP without index would lead to a partially observable MDP as the agent would not be aware of which goal in $\tau_{\mathcal{G}}$ it is targeting, which would result in unstable value learning (Sutton and Barto, 1998).

These properties are demonstrated experimentally in Figure 9.2 with ablations described in Section 9.4.4.

9.2.2 The DCIL-II algorithm

The DCIL-II algorithm first extracts the sequence of goals from a single demonstration and derives the alternative GC-MDP introduced in Section 9.2.1. By running a 2-step loop, it then learns a policy that can be used later to reach each goal and complete the complex demonstrated behavior sequentially. Algorithm 8 summarizes these different steps. The main differences with the DCIL-I algorithm presented in Algorithm 6 lie in the management of the indices and are highlighted in red.

From a single demonstration to a sequence of goals

The sequence of goals $\tau_{\mathcal{G}}$ is extracted from the demonstration exactly as in DCIL-I. We project the demonstrated states in the goal space and split the demonstration into N_{goal} sub-trajectories of equal arc lengths ϵ_{dist}. For each sub-trajectory in the goal space, we extract its final elements and concatenate them to construct $\tau_{\mathcal{G}}$. We also extract the states associated with the initial elements of each sub-trajectories. Those

Algorithm 8 DCIL-II

Input: $\pi, Q, \bar{Q}, \tau_{demo}$ ▷ actor/critic/target critic networks & demonstration
$\{g_i\}_{i \in [1, N_{goals}]} \leftarrow$ extract_goals(τ_{demo})
$B \leftarrow [\,]$ ▷ replay-buffer
$R \leftarrow \{i : [\,]\}_{i \in [1, N_{goals}]}$ ▷ successes/failures memory
for $n = 1 : N_{episode}$ **do**
  $i \leftarrow select\_index(R)$ ▷ **Trajectory initialization**
  $s \leftarrow$ env.reset(i)
  $t \leftarrow 0$
  $success, done, last\_index, timeout \leftarrow False, False, False, False$
  while not done **do** ▷ **Trajectory rollout**
    $a \sim \pi(a|s, i, g_i)$
    $s', env\_done \leftarrow env.step(a)$
    $r \leftarrow 0$
    if $s' \in \mathcal{S}_{g_i}$ **then** ▷ success
      $r, success \leftarrow 1, True$
      $last\_index \leftarrow (i \geq nb\_skills)$
      $i' \leftarrow i + 1$ ▷ index shift
      $R[i] \leftarrow R[i] + [1]$
    else if $t \geq T_{max}$ or env_done **then** ▷ failure
      $success, timeout \leftarrow False, True$
      $i' \leftarrow i$
      $R[i] \leftarrow R[i] + [0]$
    else
      $success \leftarrow False$
      $i' \leftarrow i$
    if env_done or $last\_index$ **then**
      $done \leftarrow True$
    $B \leftarrow B + (s, g_i, i, a, s', g_{i'}, i', r, done, success)$
    $s, i \leftarrow s', i'$
    $done \leftarrow done \vee timeout$
    SAC_update($\pi, Q, \bar{Q}, B$) ▷ non-modified SAC update

states form a set of demonstrated states used by DCIL-II to reset the agent in a valid success state and to train it to reach the following goal (Hypothesis 9.3).

Main Loop

DCIL-II runs a 2-step loop to train an agent and combines an off-policy actor-critic algorithm (e.g. the Soft Actor-Critic (SAC) algorithm (Haarnoja, Zhou, Abbeel, et al., 2018)) with the HER-like relabelling mechanism described in Section 9.2.1 to learn

the goal-conditioned policy.

Trajectory initialization (lines 6-9) DCIL-II selects an index i in the sequence $[1, N_{goal}]$ and the agent is reset in the associated demonstrated state.

Trajectory rollout (lines 10-30) This reset state is extended with the selected index. The policy is conditioned on the state, the selected index i and the associated goal g_i. The agent then starts a trajectory that is terminated either if the agent reaches each goal successively up to the final one, if a time limit is reached or if the agent reaches a terminal state.

This loop is repeated after the termination of each trajectory. DCIL-II keeps track of the successes and failures for each indexed trajectory (lines 18, 22 and 27). Thus, the index selection (line 6) can be biased towards indices corresponding to goals with a low ratio of success using a discrete distribution over indices that is inverse proportional to the success ratio. Such distribution is obtained using the roulette wheel selection operator commonly used in Evolutionary Algorithms (Blickle and Thiele, 1996).

The saved transitions are used to perform an actor-critic update after each step in the environment (line 31). In a sampled batch of transitions, half of the transitions are relabeled using the relabelling mechanism presented in Section 9.2.1.

9.3 Bridging the gap between DCIL-II and similar approaches

In the following paragraphs, we discuss the relations and differences between the value propagation mechanisms used in DCIL-I (see Chapter 8), option-based Hierarchical RL, R-DSC (see Section 8.3 for both) and ours.

Bridging the gap between DCIL-I & DCIL-II

In DCIL-I, instead of considering a single MDP with an extended state space, we rather considered a sequence of MDPs. In this sequence of MDPs $\mathcal{M}_i = (\mathcal{S}, \mathcal{A}, R_i, p, \gamma_i)$, only the reward function and the discount function differed from one MDP to another. First, the discount function made the success states associated to any goal terminal:

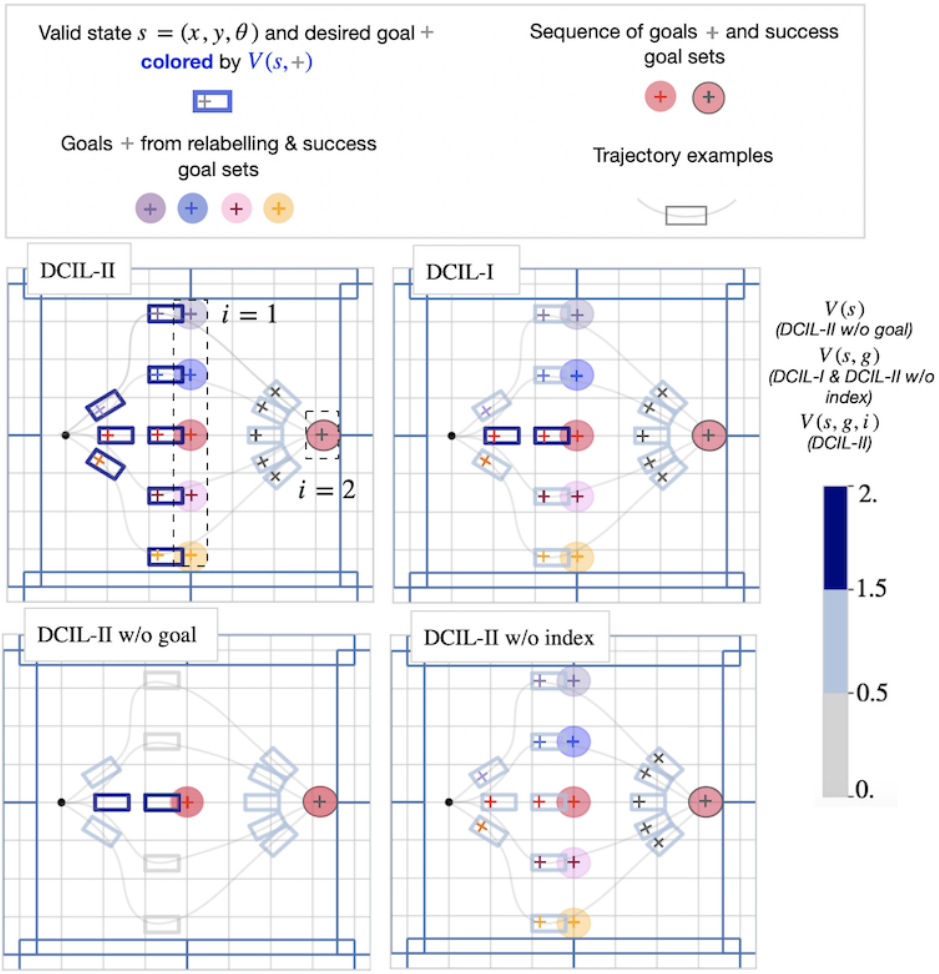


Figure 9.2: Comparison of value propagation in DCIL-II, DCIL-I, DCIL-II w/o goal and DCIL-II w/o index, using a Dubins car environment after 15.000 training steps. Unlike when success states are terminal as in DCIL-II w/o index, the value propagation mechanisms in DCIL-II, DCIL-I and DCIL-II w/o goal help propagate the value of the second goal into the value of the first $((s, g_i)$ pairs with a value ~ 2, in dark blue). Moreover, the relabelling mechanism of DCIL-II helps propagate the value associated with the second goal to the value associated with any relabeled goal (relabeled $(s, \bar{g}_i)$ pairs with a value ~ 2). In DCIL-I, the same value is propagated to the first goal in $\tau_{\mathcal{G}}$ only (relabeled $(s, \bar{g}_i)$ pairs with a value ~ 1, in light blue). Those differences between value propagation mechanisms result in the larger value of valid states leading to alternative intermediate goals in DCIL-II compared to other methods. In other words, DCIL-II helps the agent prepare for the next goal while reaching any intermediate goal.

$$\gamma_i(s) = \begin{cases} 0 \text{ if } s \in \mathcal{S}_{g_i}, \\ \gamma \text{ otherwise.} \end{cases} \tag{9.14}$$

In addition, the sparse reward received by the agent when it reached the current goal was augmented by a *reward bonus* based on the value with respect to the next MDP $V_{i+1}(s)$:

$$R_i(s_{t+1}) = \begin{cases} 1 + V_{i+1}(s_{t+1}) \text{ if } s_{t+1} \in \mathcal{S}_{g_i}, \\ 0 \text{ otherwise.} \end{cases} \tag{9.15}$$

This reward bonus was used to propagate the value from the next MDPs to the previous ones and was analogous to the non-terminal success states in DCIL-II (see Section 9.2.1). Indeed, it encouraged the agent to achieve the goal of the current MDP by reaching valid success states that were compatible with the resolution of the next MDPs. Thus, the optimal policy for one MDP in the sequence depended on the next MDP.

Using this sequence of MDPs instead of a single MDP, we avoided extending the state space to include the index of the current goal. However, as illustrated in Figure 9.2, when this framework was combined with HER, the agent only received the reward bonus when it reached the goals of the sequence but not when it reached the artificial goals created by the relabelling process. Thus, the agent was encouraged to prepare for the sequence of goals only when it was conditioned on the goals of the sequence. On the contrary, in the single MDP framework used by DCIL-II, the extended state space combined with the modified relabelling mechanism forces the agent to prepare for the next goals in the sequence while reaching both types of goals (goals of the sequence or relabeled goals).

We further underline the difference between the value propagation mechanisms of DCIL-I and DCIL-II in the ablation study conducted in Section 9.4.4.

Bridging the gap between DCIL-II & option-based HRL

We can look at the MDP $\mathcal{M}_{seq}$ (Section 9.2.1) from the perspective of the Option-based HRL framework (Precup, 2000) and see it as a semi-MDP with a hand-designed chain of options.

Indeed, we can associate an option to each goal in $\tau_{\mathcal{G}}$ by considering the set of success states of these goals as termination sets and the policy conditioned on the indices of these goals as the intra-option policies. From this perspective, we get a chain of options by considering that the termination set associated with a goal is the initiation set of the option associated with the next goal, as in (Bagaria and Konidaris, 2019).

Thus, we also have a fixed and deterministic policy over options switching to the option associated with the next goal in $\tau_{\mathcal{G}}$ as soon as the current option is completed.

We thus see that $\mathcal{M}_{seq}$ defines a restricted semi-MDP where the structure of options is a simple chain and an option can only trigger the next one.

Nevertheless, in this semi-MDP, values are propagated from one option in the chain to the previous ones via non-terminal success states. A similar semi-MDP which also propagates value between options is defined in the Option-Critic framework (Bacon et al., 2017; Klissarov et al., 2017; Harb et al., 2018). However, the two-stage learning of both the policy over options and the intra-option policies makes the end-to-end training of those policies unstable (Hutsebaut-Buysse et al., 2022). Moreover, in the absence of explicit goal-conditioning, no relabelling mechanism may be applied in $\mathcal{M}_{seq}$ or in any method based on the Option-Critic framework. This makes exploration difficult when learning how to reach sparsely rewarded goals.

By extending $\mathcal{M}_{seq}$ to GCRL (Section 9.2.1), DCIL-II can be seen as a modified Option-based method which benefits from both reward propagation and exploration mechanisms at the cost of hand-designing options.

Again, we underline the importance of combining the value propagation mechanism with an exploration mechanism by considering an ablation of DCIL-II without HER called *DCIL-II w/o goal* in Section 9.4.4.

Bridging the gap between DCIL-II & goal-conditionned skill-chaining

In DSC (Bagaria and Konidaris, 2019), the authors construct a chain of options backward from the goal. In R-DSC (Bagaria, Senthil, Slivinski, et al., 2021) which is a refined version of DSC, goal-conditioned policies are used as intra-option policies and HER is used to train them. This capability to benefit from HER makes R-DSC very close to DCIL-II. However, in R-DSC, options are not hand-designed using a sequence of goals. Instead, a learned classifier delimits the termination set of each option and a goal is sampled within this non-stationary set of states.

For each option, a set of states from which the agent has been able to achieve the termination set of the next option is constructed and updated after each episode. Within the set of goals obtained by projecting these states in the goal space, the one with the largest value is selected to condition the intra-option policy. The values of those goals are updated after each episode using the following relation:

$$V_g(p_{\mathcal{G}}(s), p_{\mathcal{G}}(s')) = \\ V_o(s, p_{\mathcal{G}}(s')) + \gamma \max_{s'' \in \epsilon_p} V_g(p_{\mathcal{G}}(s'), p_{\mathcal{G}}(s'')) \tag{9.16}$$

where V_g returns the value of goals at the level of options and V_o returns the value of states at the level of control steps.

Thus, the value is only propagated from one option to the others for the selection of goals and does not impact intra-option policies. Therefore, it is implicitly assumed that the trained intra-option aiming for the selected goal only reaches valid success states. However, if the intra-option policy changes and eventually leads the agent to invalid success states, the agent is not able to complete each option successively. Therefore, the option chain is broken until selecting another goal reached via valid success states only.

By contrast, the DCIL-II mechanism for value propagation avoids these breaks by increasing the value of valid success states and, therefore, encouraging the agent to reach each goal only via these states.

As before, we underline the importance of this value propagation mechanism by comparing DCIL-II to an ablated version which uses the same vanilla HER as R-DSC called *DCIL-II w/o index* (see Section 9.4.4).

Comparison of hypotheses

The DCIL-II algorithm relies on three main hypotheses. We assume that the agent can be reset in some states of the demonstration, that expert actions are not provided and that a definition of the goal space is given to perform GCRL. These hypotheses are summarized in Table 9.1.

Reset

The training procedure in DCIL-II assumes that the agent can be reset in some demonstrated states. A similar form of reset is assumed in DCIL-I. Stronger reset assumptions like *reset-anywhere* where the agent can be reset in uniformly sampled states have also been considered in the GCRL literature (Nasiriany et al., 2019). PWIL is based on the more classical assumption of a unique reset, and BC does not require any reset at all.

	DCIL-II	*DCIL-I*	*PWIL*	*BC*	*R-DSC*	*Option-Critic*
Learning from demonstration	•	•	•	•		
Sequential goal-reaching bias	•	•			•	
Relabelling mechanism	•	•			•	
Value propagation mechanism	•	•			•	•
Extended state space	•					•
Reset hypothesis: s_0 - single state $\{s_i\}$ - states from demo	$\{s_i\}$	$\{s_i\}$	s_0		s_0	s_0
Demonstration type: $S \times A$ - states and action S - states only	S	S	$S \times A$	$S \times A$		
Goal space definition	•	•			•	•

Table 9.1: Comparison of the problems tackled (dark grey), the mechanisms used (white) and the hypothesis made (light grey) by DCIL-II, three methods tackling the same problem of learning from a single demonstration (DCIL-I, PWIL (Dadashi et al., 2020) and BC (Pomerleau, 1991)) and two methods relying on a value propagation mechanism (R-DSC (Bagaria, Senthil, Slivinski, et al., 2021) and Option-Critic (Bacon et al., 2017)).

State-only demonstration

Similarly to DCIL-I, DCIL-II learns the desired complex behahior by leveraging a single state-based demonstration, without considering the actions. Learning from states only is crucial when the actions used in the demonstration are difficult to collect (e.g. motion capture, human guidance). On the contrary, both PWIL and BC require state-action demonstrations.

Goal-space definition

As GCRL-based methods, DCIL-II and DCIL-I require a definition of the goal space and the corresponding mapping from the state space to the goal space. No such assumption is necessary in PWIL or BC as none of them are based on GCRL.

A summary of these hypotheses is included in Table 9.1.

9.4 Experiments

In this section, we first introduce the experimental setup consisting of five environments of varying complexity. We then present the implementation details of DCIL-II and conduct an ablation study to empirically validate our design choices. Finally, we compare our method to different baselines and apply it to a more realistic simulated robotic environment.

9.4.1 Environments

We evaluate DCIL-II in five environments: the *Dubins Maze*, the *Fetch* environment (both are presented in Chapter 8), two variants of the *Humanoid* environment from the OpenAI Gym Mujoco suite (Todorov et al., 2012) (Brockman et al., 2016) and the *Cassie Run* environment [I].

Humanoid locomotion

The *Humanoid locomotion* environment is a variant of the Humanoid-v2 environment (Todorov et al., 2012; Brockman et al., 2016) with two major modifications. A state $s \in \mathbb{R}^{378}$ contains the positions of the different body parts as well as their velocities. In order to define an easily interpretable goal space corresponding to the Cartesian position of the torso, we also include the x, y position of the torso in the states, while they are generally discarded. The point is that, in order to walk to infinity, the agent must learn to ignore these two inputs which makes this variant much more difficult than the original one. To compensate for this increased difficulty, the evaluation budget is reduced to 200 control steps instead of 1000. As a result, an optimal agent covers a forward distance of ~ 10 meters instead of ~ 50 meters. Demonstrations are obtained by an agent trained with the SAC algorithm on the original dense reward associated with the environment which is much easier than our sparse reward context.

[I] code provided at https://github.com/perrin-isir/gym-cassie-run and based on the model from (Contributors, 2022)

This original reward consists of three parts: a forward reward obtained by the agent for moving forward along the x-axis, a health reward received for each step spent by the agent with its torso above 1 meter and a cost penalizing the controls and contacts. These demonstrations are divided into 15 goals by DCIL-II. Besides, the DCIL-II agent is only rewarded when the agent reaches desired goals corresponding to a desired (x, y, z) position of the torso.

Humanoid stand-up

The *Humanoid stand-up* environment is similar to the Humanoid locomotion environment. The only difference is that, instead of being initialized standing, the agent is laying down on its back at the beginning of an episode. Demonstrations are also obtained using SAC and the easier original dense reward function, which contains two parts: a height reward proportional to the position of the torso of the agent along the z-axis and a cost penalizing the controls. A demonstration corresponds to the Humanoid lifting its torso above 1.4 meters. DCIL-II divides this demonstration into 12 goals. Similarly to the Humanoid locomotion task, DCIL-II only uses a sparse reward received when the agent reaches the desired (x, y, z) position of the torso.

Cassie run

The *Cassie run* environment is a variant of the *Humanoid locomotion* environment where the humanoid is replaced by a simulated Cassie robot which recently broke a running speed record[II]. In this environment, states $s \in \mathbb{R}^{67}$ contain the Cartesian positions and the orientations of each part of the bipedal platform (including the x, y position of the robot as discussed in Section 9.4.1). A goal $g \in \mathbb{R}^3$ corresponds to the Cartesian position of the pelvis. DCIL-II only uses a sparse reward received by the agent when it reaches these low-dimensional goals. Demonstrations are obtained after $18M$ training steps using the TQC algorithm (Kuznetsov et al., 2020) on a hand-designed dense reward function (Perrin-Gilbert, 2022).

9.4.2 Baselines

We compare DCIL-II to four baselines: 1) a random initialization of the weights of the policy, used as a lower bound on the performance of an agent in each environment; 2) a naive BC method using a single demonstration; 3) PWIL, a state-of-the-art IL algorithm in the Humanoid environment with a single demonstration; 4) DCIL-I.

[II] https://today.oregonstate.edu/news/bipedal-robot-developed-oregon-state-achieves-guinness-world-record-100-meters

9.4.3 Implementation details

The hyper-parameters for each environment are presented in Table 9.2. In the three environments, the observations are normalized with the mean and standard deviation of the observations gathered during an initialization phase where random actions are sampled. Moreover, during the SAC update, the target value is clipped in $[0, N_{goals}]$. This upper bound for the value corresponds to the maximum reward that an agent can obtain along a trajectory with a discount factor $\gamma = 1$. We found that clipping the value during the critic update greatly improved the stability of the agent and limited poor generalization of the critic network. The code of DCIL-II based on the XPAG library (Perrin-Gilbert, 2022) is provided at https://github.com/AlexandreChenu/DCIL_XPAG.

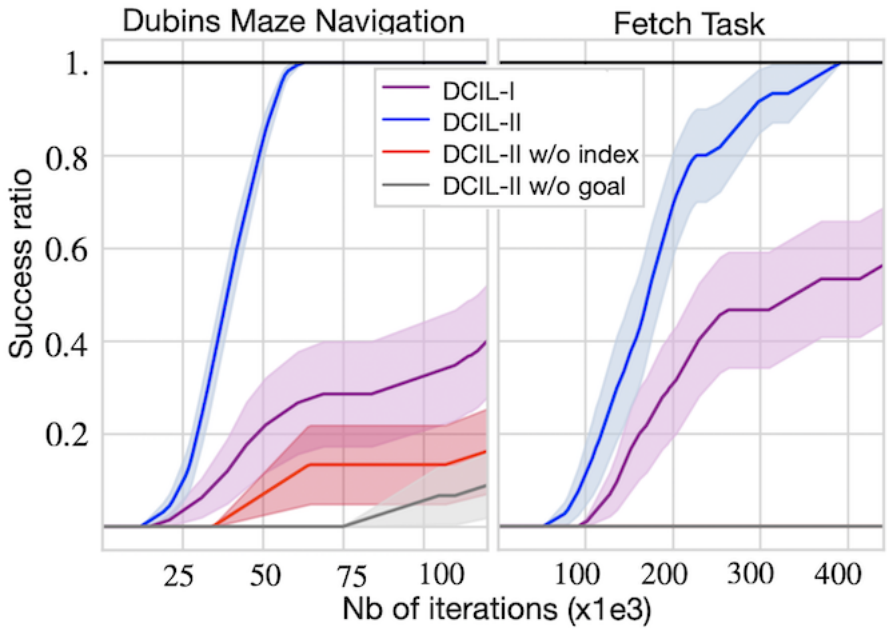


Figure 9.3: Comparing DCIL-II to three variants: we evaluate the success rates of DCIL-I, DCIL-II w/o index and DCIL-II w/o goal throughout training in the Dubins Maze and in the Fetch Task. The mean and standard deviation ranges over 15 seeds. The standard deviation is divided by four for more readability.

9.4.4 Ablation study

Using the Dubins Maze and the Fetch environments, we compare DCIL-II to three variants inspired by the discussions in Sections 9.3, 9.3 and 9.3. The first variant underlines the importance of combining the value propagation mechanism commonly used by DCIL-II and Option-Critic methods with GCRL and, in particular, a

relabelling mechanism like HER (Section 9.3). In this variant called *DCIL-II w/o goal*, the agent interacts directly with the MDP $\mathcal{M}_{seq}$ defined in Section 9.2.1. There is no explicit goal-conditioning and relabelling mechanism. Therefore, the policy is only conditioned on the index of the goals which helps propagate the value associated with a goal into the value of the previous one. However, the lack of relabelling mechanism results in poor exploration.

Similarly to R-DSC (as discussed in Section 9.3), in the second variant called *DCIL-II w/o index*, the agent interacts with a GC-MDP using a vanilla version of HER where success states are terminal. In this context, the agent benefits from efficient exploration. However, in the absence of reward propagation between successive goals, the agent is not encouraged to reach valid success states only, hampering sequential goal-reaching.

The third variant is DCIL-I. As explained in Section 9.3, the main difference between DCIL-I and DCIL-II lies in the relabelling mechanism. In DCIL-I, relabelling is as in the original version of HER and success states are terminal. The agent only receives the reward bonus used to propagate the value from goal g_i in $\tau_{\mathcal{G}}$ to the previous one g_{i-1} when it reaches this precise goal g_i, not when it reaches an artificial goal created by HER. On the contrary, in DCIL-II, artificial successful transitions created by the relabelling mechanism also propagate the value from the next indices to the previous ones.

These differences in value propagation between DCIL-II w/o index, DCIL-I and DCIL-II are illustrated in Figure 9.2. DCIL-II is able to increase the value of valid success states for any intermediate goals (original goal from $\tau_{\mathcal{G}}$ and relabeled goals). As a result, an agent trained with DCIL-II learns how to navigate the entire *Dubins Maze* and to solve the Fetch task faster than with any of the variants (see Figure 9.3).

9.4.5 Comparison to baselines

In this section, we compare DCIL-II to all baselines in the two humanoid environments.

Figure 9.4 presents the average maximum return gathered by an agent over evaluation trajectories in both environments. During the evaluation trajectory, the agent is reset in the first state of the demonstration. At each control step, actions are selected according to the mean of the action distribution returned by the policy network. The return is computed according to the original reward for each environment. However, none of the methods trained directly using this dense reward.

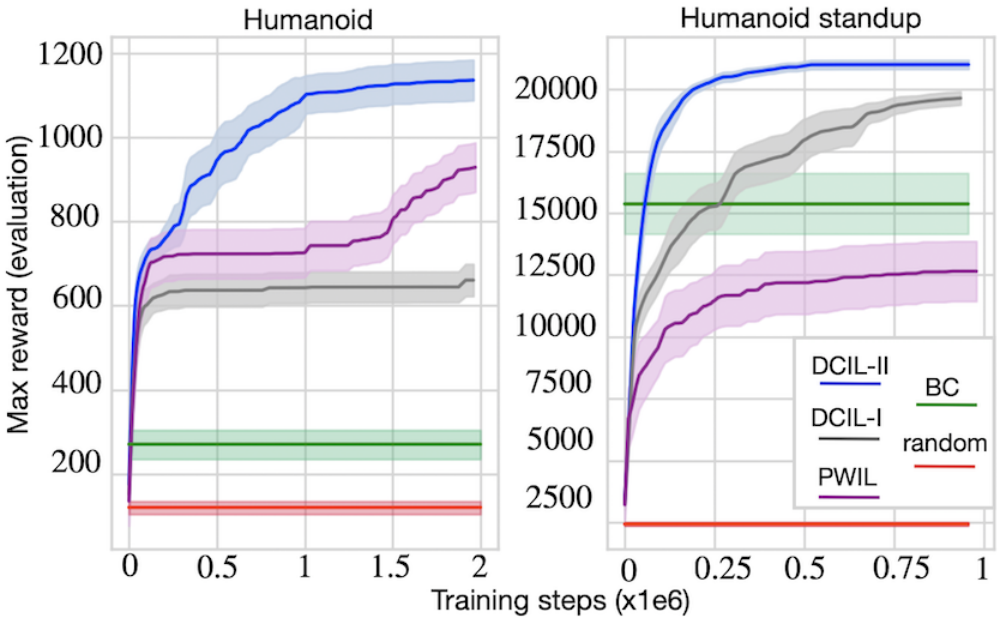


Figure 9.4: Comparing DCIL-II to a random policy, BC, PWIL and DCIL-I. We evaluate the maximum return obtained by the agent throughout training in the Humanoid locomotion and Humanoid stand-up environments. The mean and standard deviation ranges over 15 seeds. The standard deviation is divided by two for more readability.

In the locomotion task, DCIL-II is the only method able to reach an average maximum return superior to 1100 within 2M training steps. This reward corresponds to an agent walking a distance of $\sim 10m$. As discussed in the presentation of the environment, the added x, y positions of the agent forces non goal-conditioned agents (as in PWIL) to ignore these two inputs. This results in a slower increase of the return of PWIL compared to the results presented in the original paper (Dadashi et al., 2020).

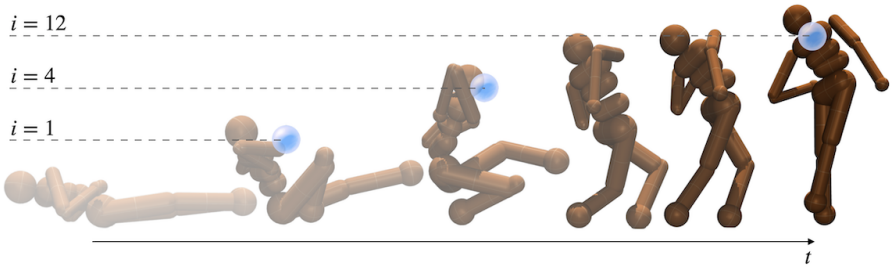


Figure 9.5: Visualization of sequential goal reaching in the *Humanoid stand-up* environment. In this environment, a sequence of 12 goals τ_G corresponding to successive Cartesian positions of the torso is used by DCIL-II to learn the behavior using only 253K training steps. Only three goals are shown here.

In the stand-up task, both DCIL-I and DCIL-II manage to achieve large returns around

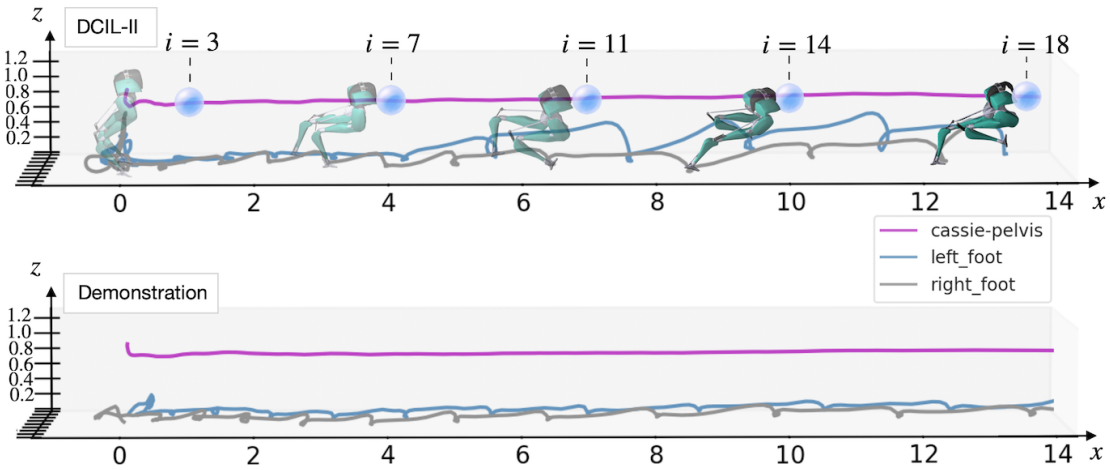


Figure 9.6: Visualization of sequential goal reaching in the *Cassie run* environment. In this environment, a sequence of 19 goals τ_G corresponding to successive Cartesian positions of the pelvis is used by DCIL-II to learn the behavior using only 707K training steps on average. Only five goals are shown here. One can see that, though the straight pelvis trajectory is well reproduced, the obtained feet trajectories differ widely from the demonstrated ones.

20.000 which correspond to an agent standing up (see Figure 9.5). Nevertheless, DCIL-II achieves such returns much faster than DCIL-I. BC performs surprisingly well in this task. This is mainly due to the fact that the entire trajectory of the agent is mostly conditioned by the very first actions. Thus, the often problematic distribution shift faced by BC (Ross and D. Bagnell, 2010) has less impact in this environment.

9.4.6 Using DCIL-II to control a simulated Cassie bipedal platform

In this environment, we want to learn a controller to make Cassie walk straight. DCIL-II manages to learn a walking behavior using only a sequence of low-dimensional goals corresponding to a succession of Cartesian positions of the pelvis. Only the position of this part of the robot is constrained during locomotion, the rest of the robot is free to follow trajectories that widely differ from the demonstration. Figure 9.6 illustrates this property. Indeed, the trajectories of the feet deviate a lot from the demonstration. In addition, at the beginning of the trajectory, after being initialized slightly above the ground in a straight position, the simulated robot learns to fall on its feet by folding and unfolding its legs. These walking and stabilization behaviors are totally different from and more complex than the demonstrated behaviors. Nevertheless, they still satisfy the desired successive positions of the pelvis. Thus, DCIL-II learns a control policy to reach 19 successive goals using $707K \pm 144K$ training steps. By reaching these goals sequentially, the system covers a forward distance of 15 meters with a stable

z-position of the pelvis.

Discussion & Conclusion

In this chapter, we leveraged a sequential inductive bias, resulting in the design of an augmented GCRL framework where the agent must successively reach low-dimensional goals in order to achieve a complex task. In this framework, the agent is encouraged to achieve each goal via states that are compatible with reaching the subsequent goals. Based on this framework, we presented the DCIL-II algorithm and we showed that it can learn control policies significantly faster than state-of-the-art imitation learning methods when addressing complex articulated behaviors with simulated and underactuated robots.

One of the main limitations of DCIL-I and DCIL-II under the perspective of applying it to a real robot is the assumption that the agent can be reset in some states selected in the demonstrated trajectory. Rather than being reset, we would like the robot to come back to a required state using its own control law, as in (Ecoffet et al., 2021; A. Gupta et al., 2021).

In the next chapter, we leverage well-known mechanisms from the Learning from Demonstration literature to propose a variant of DCIL-II called Single-Reset DCIL. This approach requires a weaker reset hypothesis and, as the name goes, only needs to reset the agent to a single state.

Hyper-parameters	*Dubins Maze*	*Fetch*	*Humanoid locomotion*	*Humanoid stand-up*	*Cassie run*
Critic hidden size	[400, 300]	[512, 512, 512]	[512, 512, 512]	[512, 512, 512]	[512, 512, 512]
Policy hidden size	[400, 300]	[512, 512, 512]	[512, 512, 512]	[512, 512, 512]	[512, 512, 512]
Activation functions	ReLU	ReLU	ReLU	ReLU	ReLU
Batch size	256	256	64	64	64
Discount factor	0.9	0.98	0.98	0.98	0.98
Entropy coefficient	1×10^{-3}	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4}
Critic lr	1×10^{-3}	1×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
Policy lr	1×10^{-3}	1×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
$\epsilon_{success}$	0.1	0.05	0.05	0.05	0.05
ϵ_{dist}	1	0.5	0.5	0.3	0.75
$\tau_{\mathcal{G}}$ size	22	12	15	12	19
Training rollout budget	25	100	100	100	100

Table 9.2: Hyper-parameters used for SAC (grey) and DCIL-II (white). The Hyper-parameters relative to SAC for *Humanoid locomotion* are extracted from (Raffin, 2018) and reused for *Humanoid stand-up* and *Cassie run*.

10 Going a step further with Single-Reset DCIL

10.1 Introduction

In the previous chapters, we introduced DCIL, a novel Deep Reinforcement Learning (DRL) algorithm that leverages a sequential bias to learn a control policy for complex robotic tasks using a single demonstration. This approach can be used to learn a goal-conditioned policy to control the system between successive intermediate low-dimensional goals. It is based on an extended Goal-Conditioned RL (GCRL) framework designed to ensure that the state resulting from reaching an intermediate goal is compatible with the achievement of the following goal. Although the approach shows unprecedented sample efficiency when applied to complex robotics tasks such as grasping or humanoid locomotion, it relies on a strong assumption that the system can be reset to some states selected in the demonstrated trajectory. This assumption limits the approach to simulated systems. Indeed, when tackling a complex robotic task such as locomotion with an under-actuated humanoid robot, one can easily imagine how difficult it would be to reset the robot to a demonstrated state corresponding to a complex configuration or including precise angular and positional velocities.

In this chapter, we propose an extension of the DCIL-II algorithm called Single-Reset DCIL (SR-DCIL). As DCIL-II, SR-DCIL learns a goal-conditioned policy to control the system between a sequence of low-dimensional goals. However, SR-DCIL is designed to work even when the strong reset assumption required by DCIL-II does not hold. Instead, we only assume that the robot can be reset to a single state at the beginning of the demonstration. To adapt DCIL-II to this more challenging setting, we call upon mechanisms inspired by the literature on Learning from Demonstrations (Atkeson and Schaal, 1997) that are likely to be compatible with the use of a single demonstration. From these mechanisms, we extract a Demo-Buffer (DB) and a Value Cloning (VC) mechanism. Both are designed to guide the agent towards compatible success

states while sequentially reaching goals. In addition, we introduce Approximate Goal Switching (AGS), a mechanism that helps the agent train for goals distant from the reset state.

This chapter is organized as follows. First, in Section 10.2, we cover related work from the RL from demonstration literature. In Section 10.3 we first highlight the capital importance of the reset hypothesis in DCIL-II. Then, to tackle the limitations induced by the weaker assumption of a system that can only be reset to a single state, we present three mechanisms (DB, VC and AGS) that can be combined with DCIL-II to yield different variants of the SR-DCIL algorithm. In Section 10.4, we evaluate these variants in several challenging robotics tasks and compare it to DCIL-II. In summary, our main contributions are: (1) we highlight the importance of the strong reset assumption made by DCIL-II; (2) under the name SR-DCIL, we propose several variants of DCIL-II capable of learning a control policy under a weaker reset assumption.

10.2 Related Work

RL and IL can be merged in different ways: offline RL, inverse RL, RL from demonstration (RLfD) (see Chapter 5 for more details). However, only a few mechanisms inspired by the literature of RLfD can be used in a complementary manner with another RL algorithm. Indeed, on the one hand, offline RL learns directly from an offline dataset of interactions and assumes no additional interaction with the environment. On the other hand, Inverse RL uses the demonstration to learn a reward function that may interfere with an already available one. On the contrary, in RLfD, one simply uses the demonstration along with additional training interactions collected by any RL agent.

Using the demonstration in RLfD can help the agent to learn more quickly and effectively at different levels. If both the demonstrated states and actions are available, the latter can guide the policy in the action selection process. Alternatively, demonstrations can be used to help the critic estimate the value function.

In the SACR2 paper (Martin et al., 2021), the authors list several variants of the Soft Actor-Critic algorithm (Haarnoja, Zhou, Abbeel, et al., 2018) augmented with different mechanisms inspired from the RLfD literature.

First, SAC Behavioral Cloning (SACBC) uses Behavioral Cloning (Pomerleau, 1991) as a regularization mechanism for the update of the actor. This is inspired by various other RLfD algorithms (A. Nair et al., 2018; Goecks et al., 2019; Fujimoto and Gu, 2021). These methods all propose two main components. First, a secondary replay buffer is

filled with transitions from demonstrations. Using these transitions, an auxiliary BC loss is computed and added to the original policy loss. However, BC usually performs poorly when only a single demonstration is available (Ross and D. Bagnell, 2010; Resnick et al., 2018; Behbahani et al., 2019). Therefore, this mechanism is not relevant to the extremely weak data regime where only a single demonstration is available.

In SAC from Demonstrations (SACfD), which is inspired by (Vecerik et al., 2017; Paine et al., 2019), a similar secondary replay buffer is filled with transitions from demonstrations. However, instead of using those transitions to compute an additional BC loss, they are directly used as training transitions during the critic update. Similarly, Soft Q Imitation Learning (SQIL) (Reddy et al., 2019) also uses demonstrated transitions for both updates. However, in SQIL, in order to encourage the agent to imitate the demonstrated behavior, the demonstrated transitions are all associated with a reward of 1 while new transitions collected by the agent receive a reward of 0. Although both SACfD and SQIL are designed to work with many demonstrations (>200 in SACfD), we show that this simple mechanism is powerful enough to efficiently guide the GCP learned by SR-DCIL.

Other approaches mixing RL with imitation-based adversarial approaches (Kang et al., 2018) or generative models (Y. Wu et al., 2021) have also been considered to perform RLfD, mostly to overcome exploration limits. However, even if these approaches significantly accelerate the underlying RL algorithms, none of them are designed to work with a single demonstration as they rely on often unstable generator-discriminator architectures (Dadashi et al., 2020).

10.3 Methods

In this section, we start by highlighting the importance of a strong reset hypothesis in the GCRL framework of DCIL-II. Then, we present two mechanisms, the Demo-Buffer and Value Cloning, designed to encourage the agent to reach valid success states only. In addition, we present a mechanism designed to help the agent to train more efficiently with this weaker reset assumption. Finally, we present the complete algorithm called Single-Reset DCIL (SR-DCIL).

10.3.1 The importance of the reset hypothesis

SR-DCIL is couched in the GCRL framework of DCIL-II (see Chapter 9). In this framework a sequence of goals $\tau_{\mathcal{G}} = \{g_i\}_{[1,N_{goal}]}$ is used to guide the agent to a difficult to achieve goal. DCIL-II learns a GCP to reach each goal in $\tau_{\mathcal{G}}$ and uses a value propaga-

tion mechanism to encourage the agent to achieve each goal by reaching valid success states for the following goal in the sequence. However, in DCIL-II, we assume that the agent can be reset to demonstrated states to learn how to reach each goal individually. If the reset assumption is weaker and the agent can only be reset to a single state, two limits arise. First, it gets difficult to train for further goals in the sequence. Moreover, the value propagation mechanism is not enough to encourage the agent to reach goals via valid success states. Those two limits are illustrated in Figure 10.1.

Training for distant goals

In the GCRL framework of DCIL-II, the goal targeted by the agent is switched to the next in $\tau_{\mathcal{G}}$ and the index is incremented only if the current goal has been achieved. If the agent is only reset to a single state, the agent must be able to sequentially reach the $i-1$ previous goals in order to begin a training rollout for goal $g_i \in \tau_{\mathcal{G}}$. Therefore, the further the goal is in the sequence, the fewer training rollouts are performed by the agent for this goal.

Moreover, using a stochastic policy in SAC makes narrow goal misses inevitable. In the context of non-holonomic environment, an unrecoverable miss of the current goal g_i is likely to occur. After missing the goal, the agent continues to aim for g_i until it is reset to its initial state, either after a time limit or after reaching a terminal state. In the meantime, no training transition for goal g_{i+1} has been collected.

On the contrary, in DCIL-II, to train for this same goal $g_i \in \tau_{\mathcal{G}}$, the agent is reset to a demonstrated state taken few steps ahead along the demonstration. Thus, the agent needs very little exploration to learn how to reach g_i as the number of control steps to reach goal g_i from the associated reset state is limited. After enough training, the agent should be able to reach any goal in $\tau_{\mathcal{G}}$ starting from the associated demonstrated reset states.

Propagating the value between successive goals

If a single reset state is available, as long as the agent has not learned how to reach goal g_i, the value of the valid and invalid success states associated with g_{i-1} should be similar. Indeed, both types of success states have a value close to one as the reward received for reaching the next goal has not yet been discovered and included in the value of valid success states. Therefore, the agent is not encouraged to achieve goal g_{i-1} by reaching valid success states. This can prevent successful training.

For instance, if the agent comes across valid success states associated with g_{i-1} often

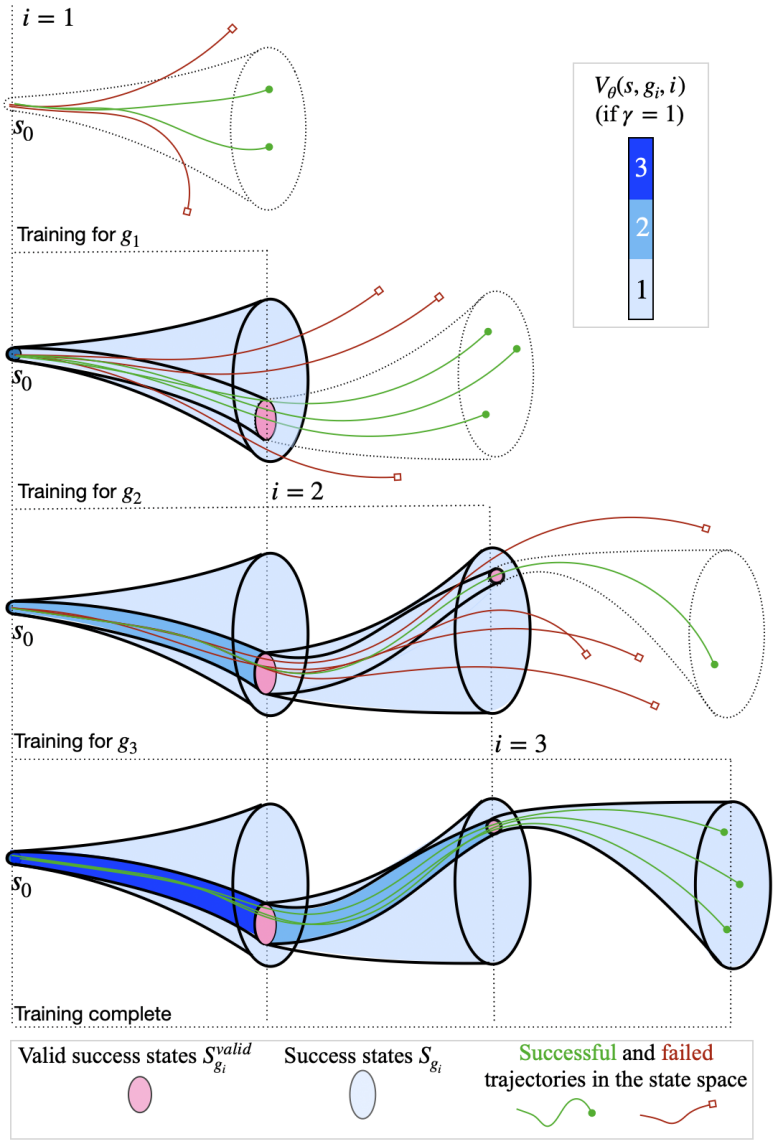


Figure 10.1: Limitations caused by a reset to a single state. The agent needs to reach g_1 to train for g_2 and to reach g_1 and g_2 to train for g_3. Moreover, to successively reach each goal, the agent must transit between successive sets of valid success states (in pink). Until the agent learns how to reach the second goal, the values of valid and invalid success states associated with the first goal are similar ($V_\theta(s, g_1, 1) = 1, \forall s \in \mathcal{S}_{g_1}$ until the agent learned how to reach g_2). Therefore, the agent is not encouraged to target valid success states. This results in a large number of wasted training trajectories (in red), as they were launched from invalid success states. When the agent mostly reaches invalid success states (as for goal g_2 here, which has a small set of valid success states), training for the next goal can become very challenging as most training rollouts for g_3 start from incompatible states.

enough by chance, after sufficient training for g_i, the reward received when reaching it increases the value of these valid states and the agent is encouraged to reach them. However, if the agent only comes across invalid success states, it is never able to reach g_i nor to propagate the value containing the associated reward. Thus the agent does not distinguish valid and invalid success states.

On the contrary, in DCIL-II, when the agent is reset to a demonstrated state to train for goal g_i, the demonstrated state corresponds to a valid success states for previous goal g_{i-1}. After enough training (i.e. successful training trajectories triggering the distance-based reward), this demonstrated valid success state should have a higher value than invalid states. Therefore, by propagating the value between the two successive goals via (9.6), the agent is encouraged to target this demonstrated valid success state as its value is higher than any other success state.

10.3.2 Using DCIL in the single-reset setting

To adapt DCIL-II to the single-reset setting, we integrate different mechanisms. Depending on whether the expert actions are available, a Demo-Buffer or Value Cloning can increase the value of valid success states. In addition, Approximated Goal Switching (AGS) can help the agent training for distant goals.

Increasing the value of valid success states

In this section, we present two mechanisms: the Demo-Buffer (DB) and the Value Cloning (VC), designed to augment the value of the valid success states. In the DB mechanism, we assume that expert actions are available, which is an additional requirement compared to the state-based demonstration necessary in DCIL-II. On the contrary, Value Cloning is an alternative to the DB mechanism if only a state-based demonstration is available.

Demo-buffer The Demo-Buffer (DB) is a secondary RB used during the actor and critic updates of SAC in the same way as SACfD (Vecerik et al., 2017; Paine et al., 2019; Martin et al., 2021). While the RB collects the training transitions, the DB is filled with the transitions extracted from the demonstration. During each SAC update, a batch of transitions is partly sampled from both buffers. The batch is filled with 80% of training transitions and 20% of transitions extracted from the demonstration. This $80-20$ ratio constitutes an additional hyper-parameter and has been chosen empirically.

The demonstration trajectory contains a successful transition to each goal leading to

the demonstrated valid success state. Therefore, by updating the critic networks using the Mean Squared Bellman Error (MSBE) (10.1) for these demonstrated transitions, the value of the state-action pairs leading to the demonstrated valid success states is increased (see Figure 10.2).

$$\mathcal{L}(\theta, \mathcal{D}_{train}, \mathcal{D}_{demo}) = \mathop{\mathbb{E}}_{(s,i,g,a,r,s',i',g') \sim \mathcal{D}_{train} \cup \mathcal{D}_{demo}} \left[\frac{1}{2} \left[Q_\theta(s,i,g,a) - (r + \gamma(s') Q_\theta(s', i', g', a)) \right]^2 \right]. \tag{10.1}$$

Thus, when reaching goal g_{i-1} in $\tau_{\mathcal{G}}$, the agent is encouraged to target the associated demonstrated valid success state. This prevents the agent from training for next goal g_i by starting from an invalid success state.

However, in order to use the DB mechanism, we make a new assumption that was not necessary in DCIL-II: the internal actions of the demonstration should be available to fill the DB.

Value cloning In Value Cloning (VC), we only assume access to the states of the demonstration. Each state in the demonstration is augmented with the associated goal and index in $\tau_{\mathcal{G}}$. This associated goal corresponds to the closest goal in $\tau_{\mathcal{G}}$ extracted from a state further along the demonstration.

Using the demonstrated trajectory, we can compute for each state the theoretical return V_{demo} received by the agent by passing through the remaining demonstrated states. This theoretical return is computed according to the reward, the transition and the discounted functions defined by the DCIL-II framework (see Section 9.2.1). It corresponds to the discounted sum of sparse rewards received for reaching each goal sequentially. The set of demonstrated states and their associated values form the Value Cloning dataset $\mathcal{D}_{VC}$.

During each SAC update, a batch of states is sampled partly from the training transitions and partly from $\mathcal{D}_{VC}$. The batch is filled with 80% of states coming from the training RB and 20% coming from $\mathcal{D}_{VC}$ similarly to DB. For states coming from the training RB, their associated target value corresponds to the on-policy soft Q-value computed with respect to the Q-value critic network Q_ω. This corresponds to a usual SAC update (see Section 3.2.4). For states extracted from $\mathcal{D}_{VC}$, the associated value corresponds to the theoretical one computed according to the demonstration.

The demonstrated states indicate a path towards valid success states. Therefore, by updating the value critic using the Mean Squared Error (MSE) between the current estimate of the value of demonstrated states and their theoretical value, their value is increased and the agent is encouraged to pass through these demonstrated states while reaching the goals in $\tau_{\mathcal{G}}$ (see Figure 10.2).

$$\mathcal{L}_{VC}(\theta, \mathcal{D}_{train}, \mathcal{D}_{VC}) = \mathop{\mathbb{E}}_{(s,i,g,V_{target})\sim\mathcal{D}_{train}\cup\mathcal{D}_{VC}} \left[\frac{1}{2}\left[V_\psi(s,i,g) - V_{target}\right]^2\right], \tag{10.2}$$

with

$$V_{target} = \begin{cases} Q(s,\pi(s),i,g) \text{ if } (s,i,g) \in \mathcal{D}_{train} \\ V_{demo} \text{ if } (s,i,g) \in \mathcal{D}_{VC} \end{cases} \tag{10.3}$$

One should note that this mechanism requires the original Actor-Critic architecture of SAC where the critic contains two networks estimating the value function and the Q-value function. In recent implementations of SAC, only the Q-value critic remains (see Section 3.2.4).

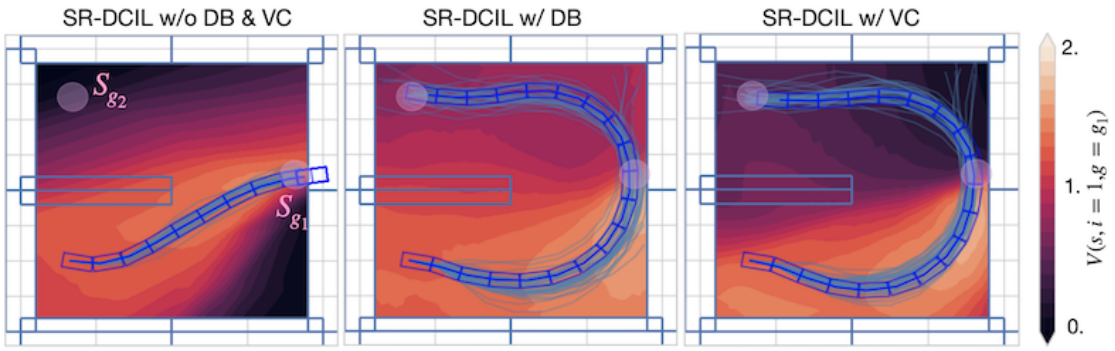


Figure 10.2: Visualising the impact of the Demo Buffer and Value Cloning after 15k training steps. When SR-DCIL is not equipped with a DB or VC (SR-DCIL w/o DB & VC), the agent is not guided to valid success states by an increase of the Q-value of demonstrated state-action pairs (as in SR-DCIL w/ DB) or an increase of the value of demonstrated states (as in SR-DCIL w/ VC). As a result, it fails to achieve g_1 via valid success states and cannot reach g_2. On the contrary, SR-DCIL w/ DB and SR-DCIL w/ VC encourage the agent to achieve g_1 by reaching valid success states and manage to reach g_2.

Increasing the number of rollouts for goals of high index

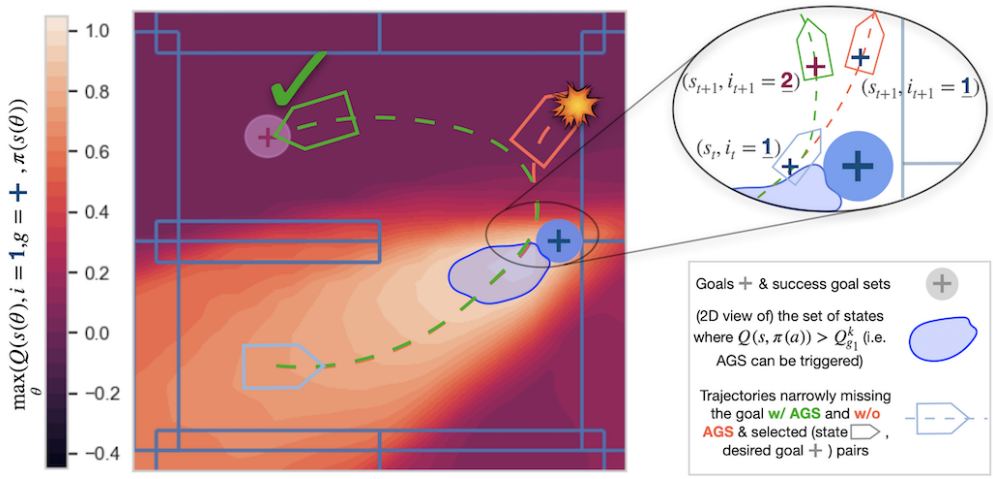


Figure 10.3: Illustration of the Approximated Goal Switching concept in a toy 2D maze where the agent corresponds to a Dubins Car (Dubins, 1957) with (x, y, θ) states and (x, y) goals. The contours represents the maximum value function obtained in the (x, y) position by uniformly sampling 20 orientations, after 15k SR-DCIL training steps. In the blue trajectory, the agent triggered AGS by entering the blue zone. k-steps after, the goal is automatically switched to the next one in $\tau_{\mathcal{G}}$, and the agent can continue its progression in the maze. On the contrary, in the green trajectory w/o AGS, after the irrecoverable narrow miss of the first goal, the agent is still conditioned on this goal and eventually collides with the wall while trying to turn toward the goal (for better readability, we drop the index and the goal in the Q-function entry when it is not necessary).

To avoid wasting a training rollout where the agent narrowly missed the goal but still managed to reach a state from which the next goal is achievable, Approximated Goal Switching (AGS) changes the current goal automatically whether the agent reaches it or misses it.

As soon as the agent reaches a state s close enough to the goal so that its Q-value $Q(s, a = \pi(s))$ is large enough, whether the agent actually reaches it or narrowly misses it a few step later, the index is automatically incremented and the current goal is switched for the next one in $\tau_{\mathcal{G}}$. Therefore, the agent starts training for the next goal in $\tau_{\mathcal{G}}$.

For each goal $g_i \in \tau_{\mathcal{G}}$, we consider that the Q-value of a state close to g_i is large enough if it is above a threshold $Q^k_{g_i}$. This threshold corresponds to the maximum Q-value associated with a state taken k-steps ahead of the success state along a successful training rollout reaching g_i. This threshold is illustrated in Figure 10.3. In each experiment of this Chapter, k is a hyper-parameter set to 2.

During the remaining k steps, the agent has enough control steps left to reach the goal. Therefore, the successful transition required to propagate the value from the next goal to the previous one in the GCRL framework of DCIL-II (see (9.6)) is still collected. Moreover, if the agent narrowly misses the goal, it can still perform a training rollout for the next goal.

The AGS mechanism avoids premature termination of training rollouts when the agent has narrowly missed a target. Therefore, it increases the number of training rollouts performed for remote goals in $\tau_{\mathcal{G}}$.

10.3.3 The SR-DCIL algorithm

Given a single demonstration, the SR-DCIL algorithm first extracts the sequence of goals and the elements required to construct the Demo-Buffer or to perform Value Cloning. The DCIL GC-MDP is derived by extending states with goals and indices exactly as in DCIL-II (see Section 9.2.1). Then, SR-DCIL runs a 2-step loop to learn a policy that can be used to reach each goal sequentially. By doing so, the agent is able to complete the complex demonstrated behavior. Algorithm 9 summarizes these different steps.

Processing the demonstration

The sequence of goals is extracted in the same way as in DCIL-I and DCIL-II. The demonstrated states are projected in the goal space and the demonstration is split into N_{goal} sub-trajectories of equal arc lengths ϵ_{dist} as in Chapters 8 and 9. For each sub-trajectory in the goal space, we extract its final elements and concatenate them to construct $\tau_{\mathcal{G}}$.

If the internal actions of the demonstration are available, the DB can be constructed using all the (state, action, next state) transitions of the demonstration. The states and next states in each transition are augmented with their associated goal in $\tau_{\mathcal{G}}$ and the corresponding index.

On the contrary, if internal actions are not available, we rely on VC to increase the value of valid success states. In that case, we construct the VC dataset $\mathcal{D}_{VC}$ by collecting all the states in the demonstration, augmenting them with their associated goal and index and calculating their theoretical discounted return.

Algorithm 9 SR-DCIL

Input: $\pi, Q, \bar{Q}, \tau_{demo}$ ▷ actor/critic/target critic networks & demonstration
$\{g_i\}_{i\in[1,N_{goals}]}, \mathcal{D}_{demo/VC} \leftarrow \text{extract}(\tau_{demo})$
$B \leftarrow [\,]$ ▷ replay-buffer
$Q_{max} = \{0\}_{i\in[1,N_{goals}]}$ ▷ Q-value thresholds (AGS)
for $n = 1 : N_{episode}$ **do** ▷ **Trajectory initialization**
    $i, T_{left}, s \leftarrow 0, T_{max}, \text{env.reset}()$
    $success, done, last\_index \leftarrow False, False, False$
    $D_{queue}, b_{AGS} \leftarrow [], \text{False}$ ▷ AGS memory + boolean
    while not done **do** ▷ **Trajectory rollout**
        $a \sim \pi(a|s, i, g_i)$
        $D_{queue}.insert((s, a, i))$
        $s', env\_done, r \leftarrow env.step(a), 0$
        if $s' \in \mathcal{S}_{g_i}$ **then** ▷ success
            $r, success \leftarrow 1, True$
            $last\_index \leftarrow (i \geq nb\_skills)$
            $i' \leftarrow i + 1$ ▷ index shift
            $Q_{max} \leftarrow update\_threshold(Q_{max}, D_{queue}, \tau_{\mathcal{G}})$
        else if $\bar{Q}(s, i, g_i, a) \geq Q_{max}$ **then**
            $T_{left} \leftarrow k$ ▷ AGS activation
            $b_{AGS} \leftarrow \text{True}$
        else if $T_{left} \leq 0$ or env_done **then** ▷ failure
            if $b_{AGS} = \text{True}$ **then** ▷ AGS switch
                $success, timeout \leftarrow False, False$
                $i' \leftarrow i + 1$
            else
                $success, timeout \leftarrow False, True$
                $i' \leftarrow 0$
        else
            $success, timeout \leftarrow False, False$
            $i' \leftarrow i$
        if env_done or $last\_index$ **then**
            $done \leftarrow True$
        $T_{left} \leftarrow T_{left} - 1$
        $B \leftarrow B + (s, g_i, i, a, s', g_{i'}, i', r, done, success)$
        $s, i \leftarrow s', i'$
        $done \leftarrow done \vee timeout$
        $\text{SAC\_update}(\pi, Q, \bar{Q}, B, \mathcal{D}_{demo/VC})$

Main loop

SR-DCIL repeatedly performs trajectory rollouts in the environment using Approximated Goal Switching to collect training transitions. It combines an off-policy actor-

critic algorithm (e.g. the Soft Actor-Critic (SAC) algorithm (Haarnoja, Zhou, Abbeel, et al., 2018)) with the HER-like relabelling mechanism of DCIL-II and DB or VC to learn the goal-conditioned policy.

Collecting transitions SR-DCIL resets the agent in the unique reset state at the beginning of the demonstration. The policy is conditioned on index 1 and the first goal g_1 in $\tau_{\mathcal{G}}$. The agent then starts a trajectory. During this trajectory, the current goal g_i is switched to the next one g_{i+1} in $\tau_{\mathcal{G}}$ if the agent reaches g_i (line 13) or if it triggered AGS (lines 19 and 22).

If the agent switches from its current goal g_i to the next one by actually reaching g_i, the associated Q-value threshold $Q^k_{g_i}$ used by AGS is potentially updated (line 17). The Q-value of the state-action pair taken k-steps ahead of the success is computed. If this value is greater than $Q^k_{g_i}$, it replaces it.

If the agent reaches each goal successively up to the final one, if it reaches a terminal state or if a time limit is reached the current trajectory is interrupted and the agent is reset to the unique reset state.

Policy update SR-DCIL performs a SAC update after each step in the environment.

If we use a DB to increase the value of valid success states, 80% of the sampled batch of transitions used to perform the actor-critic update are training transitions and 20% are demonstrated transitions from the DB. In the sampled batch of transitions, half of the transitions are relabeled using the relabelling mechanism of DCIL-II (Chenu, Serris, et al., 2022).

If we use VC instead of DB, 20% of the training batch for the value network updates contains demonstrated states. Their target values correspond to the theoretical value (see Section 10.3.2). The update of the actor remains unchanged.

10.3.4 Comparison of hypotheses

The SR-DCIL algorithm relies on three main hypotheses. First it relies on a weaker assumption compared to DCIL-II. Indeed, we assume that the agent can be reset to (or close to) the initial state of the demonstration. On the contrary, DCIL-II requires that the agent can be reset in some selected demonstrated states.

In addition, if we use a DB to increase the value of valid success states, we also assume

that expert actions are provided. Instead, if we rely on VC, this hypothesis is not formulated. Although using DB benefits from additional information and should allow the agent to learn faster than VC, learning from states only is crucial when the actions used in the demonstration are difficult to collect (e.g. motion capture, human guidance).

Finally, as a GCRL-based method, SR-DCIL requires a definition of the goal space and the corresponding mapping from the state space to the goal space.

10.4 Experiments

In this Section, we start by presenting the experimental setup. We then present an ablation study of the mechanisms designed to increase the value of valid success states and to facilitate training for advanced goals in the sequence. Finally, we compare our method to DCIL-II to assess the loss of sample efficiency induced by a weaker reset assumption.

10.4.1 Experimental setup

We evaluate SR-DCIL in three environments presented in Section 8.5.1: the *Dubins Maze* environment (Chenu, Perrin-Gilbert, and Sigaud, 2022), the *Fetch* environment (Ecoffet et al., 2021) and the *Humanoid Locomotion* environment (Todorov et al., 2012).

10.4.2 Baseline

To evaluate the drop in efficiency induced by resetting the agent to a single state, we compare SR-DCIL to DCIL-II. Indeed, by resetting the agent to demonstrated states, DCIL-II not only overcomes the limits underlined in Section 10.3.1, but it also learns a complex behavior by training on short rollouts only (see Chapter 8). Therefore, DCIL-II should be more sample efficient than RF-DCIL at learning complex behaviors.

10.4.3 Ablation study

Using the Dubins Maze and the Fetch environments, we compare five variants of SR-DCIL. Two variants benefit from the availability of demonstrated action and use DB to increase the value of valid success states. Two others assume that demonstrated actions are not available and use VC instead. In both cases, one variant (called **DB**

w/ AGS or **VC w/ AGS**) uses AGS which helps the agent train for the furthest goals in the sequence. The others (called **DB w/o AGS** or **VC w/o AGS**) do not use AGS. A final variant called **vanilla** corresponds to the application of DCIL-II in the context of a reset to a single state. This variant does not benefit from any mechanism to guide the agent toward valid success state and does not use AGS to help the agent train on distant goals.

While the *Dubins Maze* validates the utility of each mechanism, conducting a similar ablation study in *Fetch* is mandatory to evaluate the impact of high-dimensional states and action spaces on them.

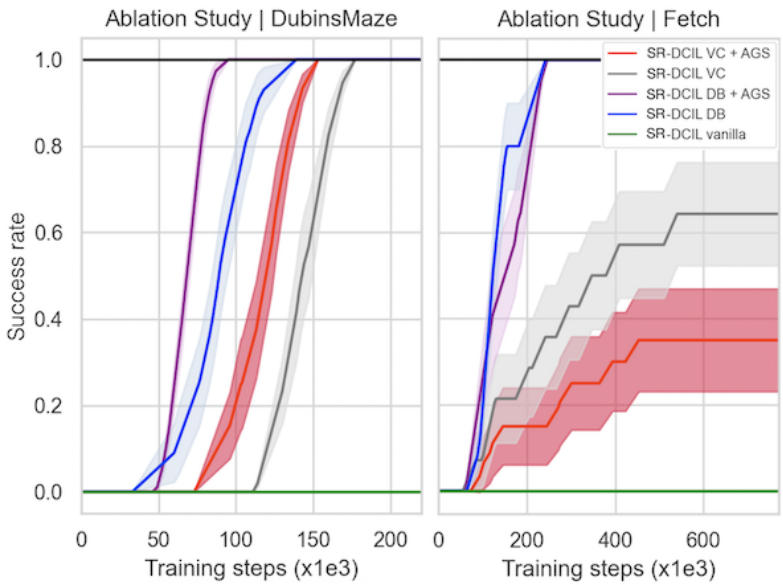


Figure 10.4: Ablation study. Comparing different variants of SR-DCIL in the Dubins Maze and the Fetch environments: we evaluate the success rates of SR-DCIL with two different mechanisms (EB and VC) to encourage the agent to reach each goal via valid success states. Both variants are evaluated with and without AGS. In addition, we evaluate a vanilla version of SR-DCIL without AGS, VC or DB equivalent with DCIL-II with a reset to a single state. The mean and standard deviation are computed over 10 seeds. The standard deviation is divided by two for better visualization.

Figure 10.4 presents the proportion of runs that solved the maze depending on the number of training steps. First, we can notice that in both environments, the variants of SR-DCIL using the DB benefit from the additional information contained in the demonstration and outperform the variants using VC.

In addition, we can notice that the AGS mechanism results in a significant gain of performance in the Dubins Maze both for SR-DCIL VC and SR-DCIL DB. However, in the Fetch environment, SR-DCIL DB w/ AGS and SR-DCIL VC w/ AGS perform worse

than their counterpart without AGS. We believe that two elements may be responsible for this difference in performance. On the one hand, the sequence of goals is shorter in Fetch compared to Dubins Maze (7 goals in Fetch compared to 17 goals in Dubins Maze). Therefore, it is easier to train for every goal in the sequence in Fetch than in Dubins Maze. On the other hand, while the Q-value threshold generalizes well in the low-dimensional state space of the Dubins Maze (see its intuitive form in Figure 10.3), it is difficult to ensure that it has an analogous form in the high-dimensional state space of Fetch. In particular, poor generalization may easily occur, resulting in an unexploitable threshold.

Finally, one should notice that, in Fetch, SR-DCIL VC only reaches a 65% success rate as, in 35% of the runs, the agent fails to achieve the first goals via valid success states and to learn how to grasp the object. Indeed, as shown in Figure 10.5, even if the Value Cloning mechanism artificially increases the value of the demonstrated states as expected, during these failed runs, the cloned value of demonstrated states has no impact on the Q-value and the policy. Thus, the agent is not guided toward valid success states.

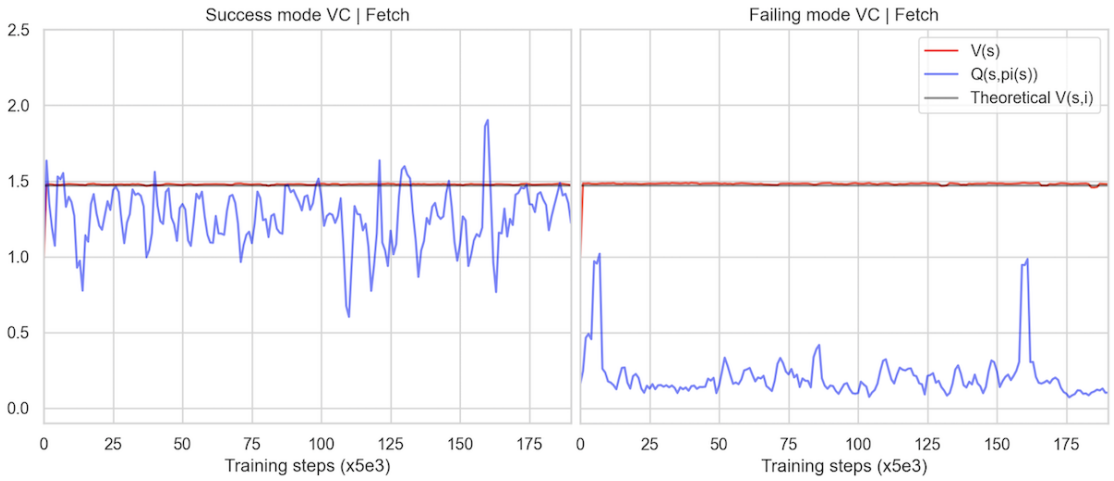


Figure 10.5: Success and failure mode of VC. In the Fetch environment, 35% of the runs fail to learn how to grasp the object. The Value Cloning mechanism sets the value of the demonstrated states (value of the last demonstrated state before grasping in red) to their theoretical value (in black) as expected. During successful runs (left panel), the agent visits states similar to the demonstrated states. Therefore, the high value is propagated in the Q-value (on-policy Q-value of the last state before grasping in blue) which impacts the policy and guides the agent toward valid success states. However, during failing runs (right panel), those states are hardly visited by the agent while training. Therefore, their high value has little to no impact on the Q-value (on-policy Q-value of the last state before grasping in blue) and the policy. As a result, the agent is not guided toward valid success states.

We believe that this absence of impact of the cloned value of demonstrated states on the Q-value results from the fact that the agent never visits states close to the demonstrated ones. Indeed, if no transition to these states is collected in the training RB, no update of the Q-value involving the cloned value can be performed. The agent is never encouraged to navigate the demonstrated states while reaching the goals.

One might think that increasing the entropy coefficient of SAC would be sufficient to encourage the agent to explore more and eventually find the demonstrated states. However, according to our observations, increasing this coefficient does not prevent these failure modes from appearing as it does not allow the agent to extensively explore the state space. For instance, the demonstrated states correspond to particularly slow approach speeds. In contrast, the agent is encouraged by the discount factor to reach each goal as quickly as possible. Thus, the optimal character of the training trajectories may prevent the agent from exploring states corresponding to slower speeds and, therefore, prevents the agent from exploring states close to those of the demonstration.

10.4.4 Comparison to the DCIL-II baseline

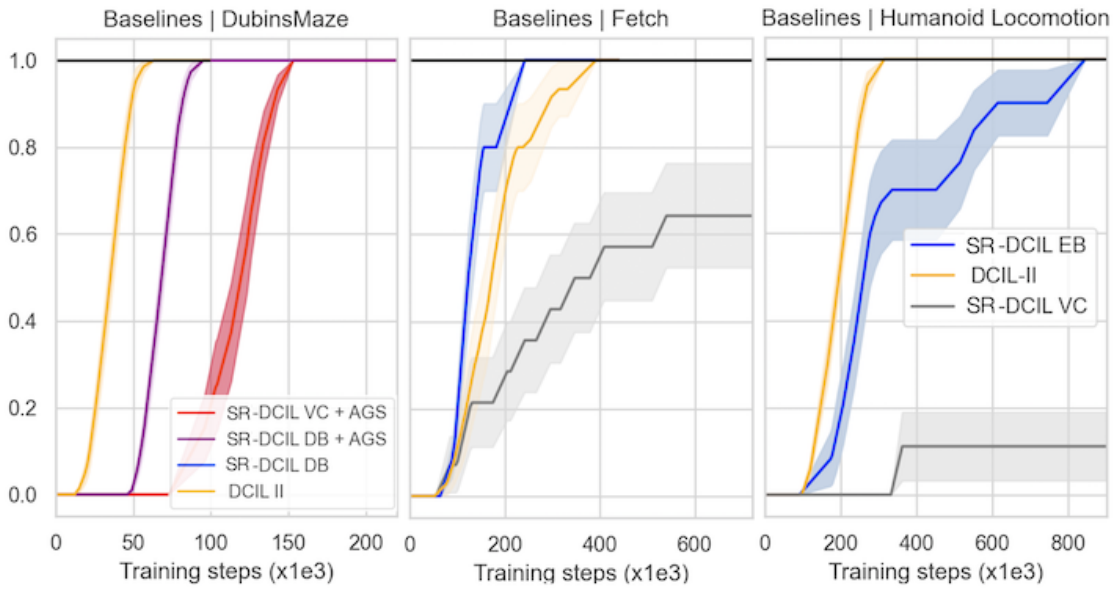


Figure 10.6: Comparison to DCIL-II. Comparing SR-DCIL EB and SR-DCIL VC in the Dubins Maze, Fetch and Humanoid Locomotion environments: we evaluate the success rates of SR-DCIL with two different mechanisms (EB and VC) to encourage the agent to reach each goal via valid success states and compare it to DCIL-II. The mean and standard deviation ranges over 10 seeds. The standard deviation is divided by two for better visualization.

In this section, we compare the best variants in each environment according to the ablation study, to DCIL-II in the Dubins Maze, the Fetch and the Humanoid Locomotion environments. Figure 10.6 presents the proportion of runs that solved each environment according to the number of training steps. Although using a weaker reset assumption, SR-DCIL DB benefits from demonstrated actions and learns how to reach goals sequentially with an efficiency close to DCIL-II. It even outperforms DCIL-II in the Fetch environment. However, the limits of VC highlighted in the ablation study are accentuated in the locomotion task. Indeed, VC solves the task only 10% of the runs.

10.5 Discussion & Conclusion

In this chapter, we have highlighted the importance of a strong reset assumption in the GCRL framework of DCIL-II. Indeed, if the agent cannot be reset to demonstrated states, the propagation of the value between successive goals is difficult. As a result, the agent cannot be guided toward valid success states and fails to learn the complex demonstrated behavior. In addition, training for distant goals while resetting the agent at the beginning of the demonstration is difficult.

We have proposed RF-DCIL, an extension of DCIL-II that includes two types of mechanisms. First, two mechanisms called Demonstration Buffer and Value Cloning are designed to encourage the agent to reach these valid success states even when we can only reset the agent at the beginning of the demonstration. Then, a third mechanism called Artificial Goal Switching is designed to help the agent train for the last goals in the sequence while always being reset at the beginning of the demonstration.

One should note that the Demonstration Buffer and the Value Cloning mechanisms do not make the same assumptions. Indeed, when using the Demonstration Buffer, we assume access to demonstrated actions (an assumption not made in DCIL-II). In that case, RF-DCIL is able to learn grasping and locomotion behaviors as efficiently as DCIL-II. On the other hand, when using Value Cloning, we suppose we do not have access to these actions (as in DCIL-II). However, Value Cloning has limitations in complex environments with high-dimensional state spaces. Similarly, while very helpful in low-dimensional state spaces, there is no benefit from using Artificial Goal Switching in high-dimensional state spaces.

RF-DCIL constitutes a first step towards applying the DCIL approach directly on physical robots. Indeed, RF-DCIL can learn a complex behavior from a single demonstration without the need for the agent to be reinitialized in demonstrated states that are potentially difficult to access (e.g. state corresponding to an unstable position of a humanoid robot). However, for the moment, only the variant requiring the demon-

strated actions currently provides satisfactory results in complex environments. For this reason, further investigation is necessary to get a Value Cloning approach that circumvents the identified failure modes.

11 Conclusion

11.1 Discussion

In this thesis, we aimed at the efficient resolution of a class of Hard Explorations Problems (HEP) corresponding to continuous control in simulated environments without exploitable objective functions. To do so, we relied on a sequential bias that we assume is present in robotic tasks to improve the efficiency of Diversity Search (DS) and Learning from Demonstration (LfD) algorithms.

First, we highlighted the non-locality of the mutation operator of DS when the policy is highly non-linear and when the environment has complex dynamics. This analysis revealed the need for more robust mutation operators in this context. We identified two critical sources of non-locality: the policy and the environment. These two sources of non-locality have been tackled separately by the DS community. Indeed, the non-locality coming from complex policies such as large neural networks has been tackled with approaches like Safe Mutations (Lehman et al., 2018; Bodnar et al., 2020). On the other hand, the non-locality coming from the complex dynamics of the environment cannot be tackled in an unknown environment. Therefore, the resulting fragility of mutations has been circumvented by the combination of DS algorithms with a learned model of the environment dynamics (Keller et al., 2020; Lim et al., 2022).

With the identified fragility of the mutation operator in mind, rather than tackling the identified source of non-locality individually, we proposed Novelty Search Skill Chaining (NSSC). With NSSC, we adopted a Divide & Conquer strategy and decomposed a complex search for diverse behaviors into simpler searches for locally diverse sub-behaviors. Despite overcoming the limits induced by the fragility of the mutation operator in holonomic environments, our experiments in a maze with non-holonomic

constraints showed that NSSC is limited in this context. Indeed, when the search for diversity is guided by low-dimensional behavior characterization, the state from which NSSC should explore locally is not fully conditioned. In certain cases, this state may be unfit for subsequent exploration. We called this state an invalid exploration state. Such states make the forward construction of a skill chain difficult, which led us to consider a different strategy to efficiently tackle HEPs.

In the second part of our contributions, we changed our approach and proposed to build on Learning from Demonstration (LfD). In order to tackle the poor sample efficiency of LfD approaches when a unique demonstration is available, we adopted again a Divide & Conquer strategy. Indeed, we decomposed the challenging task of learning a complex behavior from a single demonstration into simpler sequential goal-reaching tasks. Unlike NSSC, this approach has the advantage of learning a single (goal-conditioned) policy instead of a chain of policies.

Again, our experiments in a non-holonomic maze showed that learning a sequence of goal-reaching skills raises specific issues when the environment has a higher dimension than the goals. To circumvent these issues, we designed mechanisms to encourage the agent to reach each intermediate goal only through valid success states, i.e. states that are compatible with the achievement of the next goals. Combining the sequential goal-reaching strategy and those mechanisms resulted in the Divide & Conquer I algorithm (DCIL-I). Then, integrating these mechanisms directly in an original extended Goal Conditioned RL framework yielded a more efficient version called Divide & Conquer II (DCIL-II). We demonstrated that DCIL-II could tackle a wide class of HEPs in simulation, including complex grasping and under-actuated locomotion tasks, with unprecedented sample efficiency while learning from a single demonstration.

Overall, the successes of DCIL-I and DCIL-II highlighted the importance of targeting valid success states when reaching successive low-dimension goals in high-dimensional state space. We argue that this limitation should be considered by any approach relying on sequential goal-reaching, such as RL-based Hierarchical (Nachum et al., 2018) or Skill-Chaining methods (Bagaria, Senthil, and Konidaris, 2021; Bagaria, Senthil, Slivinski, et al., 2021). Moreover, we believe that there are two alternative options to take this limitation into account. Either the agent is explicitly encouraged to reach valid success states as in DCIL-I and DCIL-II, or the goals must contain sufficient information to avoid invalid success states. This latter approach is probably the best choice in a low-dimensional state space like the non-holonomic Dubins Maze. However, we believe this option would struggle to scale to high-dimensional state spaces as it would require powerful dimension reduction methods to avoid unreliable

distance-based sparse rewards.

Finally, the limitations of the first two DCIL approaches lied in their assumptions. Indeed, DCIL-I and DCIL-II require that the agent can be reset to any state from the demonstration. This assumption is not compatible with the long-term objective of applying these methods to physical robots. Aware of these limitations, we finally proposed several extensions to DCIL-II. One of them, RF-DCIL DB, at the cost of a stronger assumption on the content of the demonstration, presents performances close to DCIL-II in complex high-dimensional environments. The other RF-DCIL VC, although promising in low-dimensional environments, cannot yet efficiently scale to complex environments.

11.2 Conclusion & Perspective

The context of Hard Exploration Problems that we considered in this thesis corresponds to a difficult configuration for learning algorithms. Indeed, the lack of guidance caused by the absence of a well-defined objective function forces the agent to use weaker learning signals (e.g. intrinsic motivation or demonstrations). However, efficient learning of complex behaviors in this precise context is a necessary milestone in developing the future generation of robots with fast adaptive capabilities that can be deployed in unknown environments and used at a different scale than the current generation.

In this thesis, we have shown that exploiting a sequentiality bias in robotics tasks and adopting Divide & Conquer strategies can improve the efficiency of Diversity Search and Reinforcement Learning from demonstration algorithms to a certain extent.

Combining Divide & Conquer strategy with DS algorithm to construct chains of skills in a forward manner led to rather moderate results. Indeed, the limits of NSSC in non-holonomic environments make it unfit for Robot Learning applications. Nevertheless, the analysis provided in this thesis and the limits highlighted pave the way toward more robust DS algorithms based on a similar strategy. In particular, based on the identified problem of invalid exploration states in NSSC, an exciting research direction could be to combine the proposed skill-chaining-based approach with an automatic characterization of behaviors (Cully, 2019b; Paolo et al., 2020). Indeed, by including more information in a low-dimensional behavior characterization, one could boost the diversity of advanced states and, eventually, avoid falling into invalid exploration state traps.

The unprecedented sample efficiency of the DCIL-II approach opens up the path

toward robots equipped with fast complex behavior acquisition skills from a single demonstration. However, the strong reset assumption in DCIL-II still prevents its application to physical robots. This makes the improvement of single reset variants an essential future work perspective. Based on the limited but promising results of the Single-Reset variants of DCIL-II, future research could be directed toward developing robust variants that make weaker assumptions. Indeed, the weaker the assumptions, the closer we get to applying these approaches directly to physical robots. We distinguish two levels of future research objectives: short-term goals leading to a high-level, long-term goal. The first straightforward direction for future work should be to tackle the exploration problem limiting the performance of the Value Cloning mechanism in RF-DCIL (Chapter 10). Another short-term goal could be to explore the combination of DCIL-II with a state distribution matching mechanism inspired by PWIL (Dadashi et al., 2020). These two options could eventually result in a robust SR-DCIL variant requiring a single state reset and state-based demonstrations only.

However, reaching these goals would still result in an approach assuming that the agent can be reset somewhere. Despite being widely adopted in the RL community (Zhu et al., 2020), we believe this assumption is often incompatible with the concept of adaptive and autonomous robots. For instance, a humanoid robot learning to walk must be lifted back to a standing position when reset to a single state. This likely requires the intervention of a human supervisor or a hand-designed tutoring system.

Therefore, the optimal approach would be to avoid any reset. To do so, we may learn several complementary tasks instead of a single task (A. Gupta et al., 2021). For instance, returning to the humanoid robot example, one could imagine combining the locomotion task with a stand-up task. Thus the agent first learns how to stand up and then trains to walk. Each time the robot would fall, the learned stand-up task would allow it to return to a standing position autonomously. While such end-to-end learning requires considerable training costs, the DCIL approach opens the path towards fast acquisition of each skill, using a single demonstration for each.

Bibliography

Darwin, Charles (1859). *On the Origin of Species by Means of Natural Selection*. Or the Preservation of Favored Races in the Struggle for Life. Murray.

Mendel, Gregor (1865). *Experiments in plant hybridisation*.

Pearson, Karl (1901). "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin philosophical magazine and journal of science* 2.11, pp. 559–572.

Baum, Lyman Frank (1907). *Ozma of Oz: Children's Novel*. CreateSpace Independent Publishing Platform.

Collier, K. (1911). *Percy the Mechanical Man: The First Robot of Comics*.

Capek, K. (1921). *R.U.R.* Dover Thrift Editions: Plays.

Shannon, Claude Elwood (1948). "A mathematical theory of communication". In: *The Bell system technical journal* 27.3, pp. 379–423.

Asimov, Isaac (1950). *I, Robot*. Doubleday&Company, Inc.

Berlyne, Daniel E (1950). "Novelty and curiosity as determinants of exploratory behaviour". In: *British journal of psychology* 41.1, p. 68.

Bellman, Richard (1957). "A Markovian decision process". In: *Journal of mathematics and mechanics*, pp. 679–684.

Dubins, Lester E (1957). "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents". In: *American Journal of mathematics* 79.3, pp. 497–516.

Bellman, Richard (1966). "Dynamic programming". In: *Science* 153.3731, pp. 34–37.

Berlyne, Daniel E (1966). "Curiosity and Exploration: Animals spend much of their time seeking stimuli whose significance raises problems for psychology." In: *Science* 153.3731, pp. 25–33.

Bentley, Jon Louis (Sept. 1975a). "Multidimensional Binary Search Trees Used for Associative Searching". In: *Commun. ACM* 18.9, pp. 509–517. ISSN: 0001-0782.

– (1975b). "Multidimensional binary search trees used for associative searching". In: *Communications of the ACM* 18.9, pp. 509–517.

Bock, Hans Georg and Karl-Josef Plitt (1984). "A multiple shooting algorithm for direct solution of optimal control problems". In: *IFAC Proceedings Volumes* 17.2, pp. 1603–1608.

Sutton, Richard S (1988). "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1, pp. 9–44.

Garcia, Carlos E, David M Prett, and Manfred Morari (1989). "Model predictive control: Theory and practice—A survey". In: *Automatica* 25.3, pp. 335–348.

reeds, James and Lawrence Shepp (1990). "Optimal paths for a car that goes both forwards and backwards". In: *Pacific journal of mathematics* 145.2, pp. 367–393.

Whitley, Darrell, Timothy Starkweather, and Christopher Bogart (1990). "Genetic algorithms and neural networks: Optimizing connections and connectivity". In: *Parallel computing* 14.3, pp. 347–361.

Pomerleau, Dean A (1991). "Efficient training of artificial neural networks for autonomous navigation". In: *Neural computation* 3.1, pp. 88–97.

Schmidhuber, Jürgen (1991). "Curious model-building control systems". In: *Proc. international joint conference on neural networks*, pp. 1458–1463.

Dayan, Peter and Geoffrey E Hinton (1992). "Feudal Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by S. Hanson, J. Cowan, and C. Giles. Vol. 5. Morgan-Kaufmann.

Watkins, Christopher JCH and Peter Dayan (1992). "Q-learning". In: *Machine learning* 8.3, pp. 279–292.

Kaelbling, Leslie Pack (1993). "Learning to Achieve Goals". In: *IN PrOC. OF IJCAI-93*. Morgan Kaufmann, pp. 1094–1098.

Floreano, Dario and Francesco Mondada (1994). "Automatic creation of an autonomous agent: Genetic evolution of a neural network driven robot". In: *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*. The MIT Press, pp. 421–430.

Blickle, Tobias and Lothar Thiele (Dec. 1996). "A Comparison of Selection Schemes Used in Evolutionary Algorithms". In: *Evolutionary Computation* 4.4, pp. 361–394. ISSN: 1063-6560. DOI: 10.1162/evco.1996.4.4.361. eprint: https://direct.mit.edu/evco/article-pdf/4/4/361/1492921/evco.1996.4.4.361.pdf. URL: https://doi.org/10.1162/evco.1996.4.4.361.

Jain, Anil K, Jianchang Mao, and K Moidin Mohiuddin (1996). "Artificial neural networks: A tutorial". In: *Computer* 29.3, pp. 31–44.

Schaal, Stefan (1996). "Learning from demonstration". In: *Advances in neural information processing systems* 9.

Atkeson, Christopher G. and Stefan Schaal (1997). "robot learning from demonstration". In: *ICML*. Vol. 97, pp. 12–20.

Hsu, D., J. -. Latombe, and R. Motwani (1997). "Path planning in expansive configuration spaces". In: *Proceedings of International Conference on Robotics and Automation*. Vol. 3, 2719–2726 vol.3.

Knuth, Donald Ervin (1997). *The art of computer programming*. Vol. 3. Pearson Education.

Mitchell, Tom M (1997). *Machine learning*. Vol. 1. 9. McGraw-hill New York.

Hsu, David et al. (1998). "On Finding Narrow Passages with Probabilistic Roadmap Planners". In: *Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics on Robotics: The Algorithmic Perspective: The Algorithmic Perspective*. WAFr 98. Houston, Texas, USA: A. K. Peters, Ltd., pp. 141–153. ISBN: 1568810814.

LaValle, Steven M et al. (1998). "rapidly-exploring random trees: A new tool for path planning". In: *The annual research report*.

Lavalle, Steven M. (1998). *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Tech. rep. 98-11. Computer Science Department, Iowa State University.

Russell, Stuart (1998). "Learning agents for uncertain environments". In: *Proceedings of the eleventh annual conference on Computational learning theory*, pp. 101–103.

Sutton, Richard S and Andrew Barto (1998). *reinforcement learning: an Introduction*. MIT press Cambridge.

Burl, Jeff B and Jeffrey B Burl (1999). *Linear optimal control*. Addison-Wesley [1999].

Moore, Andrew W., Leemon C. Baird, and Leslie Pack Kaelbling (1999). "Multi-Value-Functions: Efficient Automatic Action Hierarchies for Multiple Goal MDPs". In: *IJCAI*.

Schaal, Stefan (1999). "Is imitation learning the route to humanoid robots?" In: *Trends in cognitive sciences* 3.6, pp. 233–242.

Sutton, Richard S, Doina Precup, and Satinder Singh (1999). "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial intelligence* 112.1-2, pp. 181–211.

Precup, Doina (2000). *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst.

Aggarwal, Charu C, Alexander Hinneburg, and Daniel A Keim (2001). "On the surprising behavior of distance metrics in high dimensional space". In: *International conference on database theory*. Springer, pp. 420–434.

Gopnik, Alison, Andrew Meltzoff, and Patricia Kuhl (Mar. 2001). *The Scientist in the Crib: Minds, Brains and How Children Learn*. Vol. 189. ISBN: 0-688-17788-3. DOI: 10.1097/00005053-200103000-00011.

Brafman, Ronen I and Moshe Tennenholtz (2002). "r-max-a general polynomial time algorithm for near-optimal reinforcement learning". In: *Journal of Machine Learning Research* 3.Oct, pp. 213–231.

Deb, Kalyanmoy et al. (2002). "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE transactions on evolutionary computation* 6.2, pp. 182–197.

Kearns, Michael and Satinder Singh (2002). "Near-optimal reinforcement learning in polynomial time". In: *Machine learning* 49.2, pp. 209–232.

Stanley, Kenneth O and Risto Miikkulainen (2002). "Evolving neural networks through augmenting topologies". In: *Evolutionary computation* 10.2, pp. 99–127.

Eiben, Agoston E, James E Smith, et al. (2003). *Introduction to evolutionary computing*. Vol. 53. Springer.

Uicker, John Joseph et al. (2003). *Theory of machines and mechanisms*. Vol. 768. Oxford University Press New York.

Kajita, Shuuji et al. (2004). "Biped walking on a low friction floor". In: *2004 IEEE/rSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 4. IEEE, pp. 3546–3552.

Ashlock, Daniel (2006). *Evolutionary computation for modeling and optimization*. Vol. 571. Springer.

Diehl, Moritz et al. (2006). "Fast direct multiple shooting algorithms for optimal robot control". In: *Fast motions in biomechanics and robotics*. Springer, pp. 65–93.

LaValle, Steven M (2006). *Planning algorithms*. Cambridge university press.

Calinon, Sylvain and Aude Billard (2007). "Incremental learning of gestures by imitation in a humanoid robot". In: pp. 255–262.

Peters, Jan and Stefan Schaal (2007). "Reinforcement learning by reward-weighted regression for operational space control". In: *Proceedings of the 24th international conference on Machine learning*, pp. 745–750.

Billard, Aude et al. (2008). "robot programming by demonstration". In: *Springer handbook of robotics*. Springer, pp. 1371–1394.

Ziebart, Brian D et al. (2008). "Maximum entropy inverse reinforcement learning." In: *Aaai*. Vol. 8. Chicago, IL, USA, pp. 1433–1438.

Hastie, Trevor et al. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer.

Konidaris, George and Andrew Barto (2009). "Skill discovery in continuous reinforcement learning domains using skill chaining". In: *Advances in neural information processing systems* 22.

Stanley, Kenneth O, David B D'Ambrosio, and Jason Gauci (2009). "A hypercube-based encoding for evolving large-scale neural networks". In: *Artificial life* 15.2, pp. 185–212.

Calinon, Sylvain et al. (2010). "Learning and reproduction of gestures by imitation". In: *IEEE Robotics & Automation Magazine* 17.2, pp. 44–54.

Konidaris, George et al. (2010). "Constructing Skill Trees for Reinforcement Learning Agents from Demonstration Trajectories". In: *Advances in Neural Information Pro-*

cessing Systems. Ed. by J. Lafferty et al. Vol. 23. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2010/file/27ed0fb950b856b06e1273989422e7d3-Paper.pdf.

Ross, Stéphane and Drew Bagnell (May 2010). "Efficient Reductions for Imitation Learning". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna resort, Sardinia, Italy: PMLR, pp. 661–668. URL: https://proceedings.mlr.press/v9/ross10a.html.

Cengel, Yunus A, Michael A Boles, and Mehmet Kanoğlu (2011). *Thermodynamics: an engineering approach*. Vol. 5. McGraw-hill New York.

Dalibard, Sébastien and Jean-Paul Laumond (2011). "Linear dimensionality reduction in random motion planning". In: *The International Journal of Robotics Research* 30.12, pp. 1461–1476.

Kober, Jens and Jan Peters (2011). "Policy search for motor primitives in robotics". In: *Machine learning* 84.1, pp. 171–203.

Lehman, Joel and Kenneth O Stanley (June 2011a). "Abandoning Objectives: Evolution Through the Search for Novelty Alone". en. In: *Evolutionary Computation* 19.2, pp. 189–223. ISSN: 1063-6560, 1530-9304. (Visited on 02/18/2020).

– (2011b). "Improving evolvability through novelty search and self-adaptation". In: *2011 IEEE congress of evolutionary computation (CEC)*. IEEE, pp. 2693–2700.

Levy, Kfir Y and Nahum Shimkin (2011). "Unified inter and intra options learning using policy gradient methods". In: *European Workshop on Reinforcement Learning*. Springer, pp. 153–164.

Ross, Stéphane, Geoffrey Gordon, and Drew Bagnell (2011). "A reduction of imitation learning and structured prediction to no-regret online learning". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLr Workshop and Conference Proceedings, pp. 627–635.

Awrejcewicz, Jan (2012). "Statics and Dynamics in Generalized Coordinates". In: *Classical Mechanics*. Springer, pp. 107–206.

Connell, Jonathan H and Sridhar Mahadevan (2012). *robot learning*. Vol. 233. Springer Science & Business Media.

Curcio, Bruna Da Rosa and Carmo Nogueira (2012). "Newborn adaptations and healthcare throughout the first age of the foal". In: *Animal Reproduction* 9, pp. 182–187.

Latombe, Jean-Claude (2012). *robot motion planning*. Vol. 124. Springer Science & Business Media.

Mason, Matthew T (2012). "Creation myths: The beginnings of robotics research". In: *IEEE robotics & automation magazine* 19.2, pp. 72–77.

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/rSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

Woolley, Brian G and Kenneth O Stanley (2012). "Exploring Promising Stepping Stones by Combining Novelty Search with Interactive Evolution". In: *Corr* abs/1207.6682. arXiv: 1207.6682.

Bongard, Josh C (2013). "Evolutionary Robotics". In: *Communications of the ACM* 56.8, pp. 74–83.

Kingma, Diederik P and Max Welling (2013). "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114*.

Kober, Jens, J Andrew Bagnell, and Jan Peters (2013). "reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11, pp. 1238–1274.

Mnih, Volodymyr et al. (2013). "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602*.

Tanedo, Flip (2013). "Notes on non-holonomic constraints". In: *P3318: Analytical Mechanics*.

Deb, Kalyanmoy and Debayan Deb (Feb. 2014). "Analysing mutation schemes for real-parameter genetic algorithms". In: *International Journal of Artificial Intelligence and Soft Computing* 4, pp. 1–28.

Goodfellow, Ian J. et al. (2014). *Generative Adversarial Networks*. arXiv: 1406.2661 [stat.ML].

Gutierrez, Juan Manuel Parrilla et al. (2014). "Evolution of oil droplets in a chemorobotic platform". In: *Nature communications* 5.1, pp. 1–8.

Atkeson, Christopher G. et al. (2015). "No falls, no resets: reliable humanoid behavior in the DARPA robotics challenge". In: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pp. 623–630. DOI: 10.1109/HUMANOIDS.2015.7363436.

Cully, Antoine et al. (2015). "robots that can adapt like animals". In: *Nature* 521.7553, pp. 503–507.

Doncieux, Stephane et al. (2015). "Evolutionary Robotics: What, Why, and Where to". In: *Frontiers in Robotics and AI* 2. ISSN: 2296-9144. DOI: 10.3389/frobt.2015.00004. URL: https://www.frontiersin.org/articles/10.3389/frobt.2015.00004.

Eiben, Agoston E and James E Smith (2015). "From evolutionary computation to the evolution of things". In: *Nature* 521.7553, pp. 476–482.

Kober, Jens, Michael Gienger, and Jochen J Steil (2015). "Learning movement primitives for force interaction tasks". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3192–3199.

Lillicrap, Timothy P et al. (2015). "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971*.

Mouret, Jean-Baptiste and Jeff Clune (2015). "Illuminating search spaces by mapping elites". In: *arXiv preprint arXiv:1504.04909*.

Neumann, Klaus and Jochen J Steil (2015). "Learning robot motions with stable dynamical systems under diffeomorphic transformations". In: *robotics and Autonomous Systems* 70, pp. 1–15.

Schaul, Tom et al. (2015). "Universal value function approximators". In: *International conference on machine learning*. PMLR, pp. 1312–1320.

Schulman, John et al. (2015). "Trust region policy optimization". In: *International conference on machine learning*. PMLR, pp. 1889–1897.

Stadie, Bradly C, Sergey Levine, and Pieter Abbeel (2015). "Incentivizing exploration in reinforcement learning with deep predictive models". In: *arXiv preprint arXiv:1507.00814*.

Bellemare, Marc et al. (2016). "Unifying count-based exploration and intrinsic motivation". In: *Advances in neural information processing systems* 29.

Benureau, Fabien and Pierre-Yves Oudeyer (Mar. 2016). "Behavioral Diversity Generation in Autonomous Exploration through reuse of Past Experience". en. In: *Frontiers in Robotics and AI* 3, pp. 1–2. ISSN: 2296-9144. (Visited on 02/18/2020).

Brockman, Greg et al. (2016). "Openai gym". In: *arXiv preprint arXiv:1606.01540*.

Forestier, Sebastien and Pierre-Yves Oudeyer (Oct. 2016). "Modular active curiosity-driven discovery of tool use". en. In: *2016 IEEE/rSJ International Conference on Intelligent robots and Systems (IROS)*. Daejeon, South Korea: IEEE, pp. 3965–3972. ISBN: 978-1-5090-3762-9. (Visited on 02/18/2020).

Goodfellow, Ian J., Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. MIT press.

Gregor, Karol, Danilo Jimenez Rezende, and Daan Wierstra (2016). "Variational intrinsic control". In: *arXiv preprint arXiv:1611.07507*.

Ho, Jonathan and Stefano Ermon (2016a). "Generative Adversarial Imitation Learning". In: *Corr* abs/1606.03476. arXiv: 1606.03476. URL: http://arxiv.org/abs/1606.03476.

– (2016b). "Generative adversarial imitation learning". In: *Advances in neural information processing systems* 29.

Houthooft, Rein et al. (2016). "Vime: Variational information maximizing exploration". In: *Advances in neural information processing systems* 29.

Oudeyer, Pierre-Yves and Linda Smith (Mar. 2016). "How Evolution May Work Through Curiosity-Driven Developmental Process". In: *Topics in cognitive science* 8. DOI: 10.1111/tops.12196.

Perrin, Nicolas and Philipp Schlehuber-Caissier (2016). "Fast diffeomorphic matching to learn globally asymptotically stable nonlinear dynamical systems". In: *Systems & Control Letters* 96, pp. 51–59.

Peters, Jan et al. (2016). “robot learning”. In: *Springer Handbook of Robotics*. Springer, pp. 357–398.

Silver, David et al. (2016). “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587, pp. 484–489.

Andrychowicz, Marcin et al. (2017). “Hindsight experience replay”. In: *arXiv preprint arXiv:1707.01495*.

Arjovsky, Martin, Soumith Chintala, and Léon Bottou (2017). “Wasserstein generative adversarial networks”. In: *International conference on machine learning*. PMLR, pp. 214–223.

Arulkumaran, Kai et al. (2017). “Deep reinforcement learning: A brief survey”. In: *IEEE Signal Processing Magazine* 34.6, pp. 26–38.

Bacon, Pierre-Luc, Jean Harb, and Doina Precup (2017). “The option-critic architecture”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1.

Florensa, Carlos et al. (2017). “reverse curriculum generation for reinforcement learning”. In: *Conference on robot learning*. PMLR, pp. 482–495.

Gangwani, Tanmay and Jian Peng (2017). “Policy optimization by genetic distillation”. In: *arXiv preprint arXiv:1711.01012*.

Hutter, Marco et al. (2017). “Anymal-toward legged robots for harsh environments”. In: *Advanced Robotics* 31.17, pp. 918–931.

Klissarov, Martin et al. (2017). “Learnings options end-to-end for continuous action tasks”. In: *arXiv preprint arXiv:1712.00004*.

Laschi, Cecilia et al. (2017). *Soft Robotics: Trends, Applications and Challenges*. Vol. 17. Springer.

Pathak, Deepak et al. (2017). “Curiosity-driven exploration by self-supervised prediction”. In: *International conference on machine learning*. PMLR, pp. 2778–2787.

Polydoros, Athanasios S and Lazaros Nalpantidis (2017). “Survey of model-based reinforcement learning: Applications on robotics”. In: *Journal of Intelligent & Robotic Systems* 86.2, pp. 153–173.

Salimans, Tim et al. (2017). “Evolution strategies as a scalable alternative to reinforcement learning”. In: *arXiv preprint arXiv:1703.03864*.

Schulman, John et al. (2017). “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347*.

Silver, David et al. (2017). “Mastering the game of Go without human knowledge”. In: *Nature* 550.7676, pp. 354–359.

Stadie, Bradly C, Pieter Abbeel, and Ilya Sutskever (2017). “Third-person imitation learning”. In: *arXiv preprint arXiv:1703.01703*.

Such, Felipe Petroski et al. (2017). "Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning". In: *Corr* abs/1712.06567, pp. 1–2. arXiv: 1712.06567.

Tang, Haoran et al. (2017). "# exploration: A study of count-based exploration for deep reinforcement learning". In: *Advances in neural information processing systems* 30.

Vecerik, Mel et al. (2017). "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards". In: *arXiv preprint arXiv:1707.08817*.

Burda, Yuri, Harrison Edwards, Deepak Pathak, et al. (2018). "Large-scale study of curiosity-driven learning". In: *arXiv preprint arXiv:1808.04355*.

Burda, Yuri, Harrison Edwards, Amos Storkey, et al. (2018). "Exploration by random network distillation". In: *arXiv preprint arXiv:1810.12894*.

Colas, Cédric, Olivier Sigaud, and Pierre-Yves Oudeyer (2018). "GEP-PG: Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms". In: *Corr* abs/1802.05054. arXiv: 1802.05054.

Cully, Antoine and Yiannis Demiris (2018a). "Hierarchical behavioral repertoires with unsupervised descriptors". In: *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 69–76.

– (2018b). "Quality and Diversity Optimization: A Unifying Modular Framework". In: *IEEE Transactions on Evolutionary Computation* 22.2, pp. 245–259.

Doncieux, Stephane et al. (2018). "Open-ended learning: a conceptual framework based on representational redescription". In: *Frontiers in neurorobotics* 12, p. 59.

Eysenbach, Benjamin et al. (2018). "Diversity is all you need: Learning skills without a reward function". In: *arXiv preprint arXiv:1802.06070*.

Faust, Aleksandra et al. (2018). "Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5113–5120.

Florensa, Carlos et al. (2018). "Automatic goal generation for reinforcement learning agents". In: *International conference on machine learning*. PMLR, pp. 1515–1528.

Fujimoto, Scott, Herke Hoof, and David Meger (2018). "Addressing function approximation error in actor-critic methods". In: *International Conference on Machine Learning*. PMLR, pp. 1587–1596.

Gottlieb, Jacqueline and Pierre-Yves Oudeyer (2018). "Towards a neuroscience of active sampling and curiosity". In: *Nature Reviews Neuroscience* 19.12, pp. 758–770.

Griffiths, David J and Darrell F Schroeter (2018). *Introduction to quantum mechanics*. Cambridge university press.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, et al. (2018). "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR, pp. 1861–1870.

Haarnoja, Tuomas, Aurick Zhou, Kristian Hartikainen, et al. (2018). "Soft actor-critic algorithms and applications". In: *arXiv preprint arXiv:1812.05905*.

Harb, Jean et al. (2018). "When waiting is not an option: Learning options with a deliberation cost". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1.

Henderson, Peter et al. (2018). "OptionGAN: Learning joint reward-policy options using generative adversarial inverse reinforcement learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32.

Kang, Bingyi, Zequn Jie, and Jiashi Feng (2018). "Policy optimization with demonstrations". In: *International conference on machine learning*. PMLR, pp. 2469–2478.

Kleinbort, Michal et al. (Sept. 2018). "Probabilistic completeness of RRT for geometric and kinodynamic planning with forward propagation". en. In: *arXiv:1809.07051 [cs]*. arXiv: 1809.07051. (Visited on 02/20/2020).

Kostrikov, Ilya, Kumar Krishna Agrawal, Debidatta Dwibedi, et al. (2018). "Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning". In: *arXiv preprint arXiv:1809.02925*.

Kostrikov, Ilya, Kumar Krishna Agrawal, Sergey Levine, et al. (2018). "Addressing Sample Inefficiency and Reward Bias in Inverse Reinforcement Learning". In: *Corr* abs/1809.02925. arXiv: 1809.02925. URL: http://arxiv.org/abs/1809.02925.

Krotkov, Eric et al. (2018). "The DARPA Robotics Challenge Finals: Results and Perspectives". In: *The DARPA Robotics Challenge Finals: Humanoid Robots To The Rescue*. Ed. by Matthew Spenko, Stephen Buerger, and Karl Iagnemma. Cham: Springer International Publishing, pp. 1–26. ISBN: 978-3-319-74666-1. DOI: 10.1007/978-3-319-74666-1_1. URL: https://doi.org/10.1007/978-3-319-74666-1_1.

Lehman, Joel et al. (May 2018). "Safe Mutations for Deep and recurrent Neural Networks through Output Gradients". en. In: *arXiv:1712.06563 [cs]*. arXiv: 1712.06563. (Visited on 03/09/2021).

Marsh, Allison (2018). *Elektro the Moto-Man Had the Biggest Brain at the 1939 World's Fair*. URL: https://spectrum.ieee.org/amp/elektro-the-motoman-had-the-biggest-brain-at-the-1939-worlds-fair-2650277507.

Merel, Josh et al. (2018). "Hierarchical visuomotor control of humanoids". In: *arXiv preprint arXiv:1811.09656*.

Nachum, Ofir et al. (2018). "Data-efficient hierarchical Reinforcement learning". In: *Advances in neural information processing systems* 31.

Nair, Ashvin et al. (2018). "Overcoming exploration in reinforcement learning with demonstrations". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 6292–6299.

Nair, Ashvin V et al. (2018). "Visual reinforcement learning with imagined goals". In: *Advances in neural information processing systems* 31.

Peng, Xue Bin et al. (2018). “Deepmimic: Example-guided deep reinforcement learning of physics-based character skills”. In: *ACM Transactions On Graphics (TOG)* 37.4, pp. 1–14.

Pourchot, Aloïs and Olivier Sigaud (2018). “CEM-rL: Combining evolutionary and gradient-based methods for policy search”. In: *Corr* abs/1810.01222. arXiv: 1810.01222.

Raffin, Antonin (2018). *rL Baselines Zoo*. https://github.com/araffin/rl-baselines-zoo.

Resnick, Cinjon et al. (2018). “Backplay:" Man muss immer umkehren"”. In: *arXiv preprint arXiv:1807.06919*.

Salimans, Tim and Richard Chen (2018). “Learning montezuma’s revenge from a single demonstration”. In: *arXiv preprint arXiv:1812.03381*.

Sermanet, Pierre et al. (2018). “Time-contrastive networks: Self-supervised learning from video”. In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 1134–1141.

Silver, David et al. (2018). “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362.6419, pp. 1140–1144.

Silvério, Joao et al. (2018). “Probabilistic learning of torque controllers from kinematic and force constraints”. In: *2018 IEEE/rSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 1–8.

Such Petroski, Felipe et al. (Apr. 2018). “Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for reinforcement Learning”. en. In: *arXiv:1712.06567 [cs]*. arXiv: 1712.06567. (Visited on 02/25/2020).

Akkaya, Ilge et al. (2019). “Solving rubik’s cube with a robot hand”. In: *arXiv preprint arXiv:1910.07113*.

Aubret, Arthur, Laetitia Matignon, and Salima Hassas (2019). “A survey on intrinsic motivation in reinforcement learning”. In: *arXiv preprint arXiv:1908.06976*.

Bagaria, Akhil and George Konidaris (2019). “Option discovery using deep skill chaining”. In: *International Conference on Learning Representations*.

Behbahani, Feryal et al. (2019). “Learning from demonstration in the wild”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 775–781.

Colas, Cédric, Olivier Sigaud, and Pierre-Yves Oudeyer (2019). “A Hitchhiker’s Guide to Statistical Comparisons of Reinforcement Learning Algorithms”. In: *arXiv preprint arXiv:1904.06979*.

Cully, Antoine (2019a). “Autonomous skill discovery with Quality-Diversity and Unsupervised Descriptors”. In: *Corr* abs/1905.11874. arXiv: 1905.11874.

– (2019b). “Autonomous skill discovery with quality-diversity and unsupervised descriptors”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 81–89.

Ding, Yiming et al. (2019). "Goal-conditioned Imitation Learning". In: *Corr* abs/1906.05838. arXiv: 1906.05838. URL: http://arxiv.org/abs/1906.05838.

Ecoffet, Adrien et al. (2019a). *Go-Explore: a New Approach for Hard-Exploration Problems*. arXiv: 1901.10995 [cs.LG].

– (2019b). "Go-explore: a new approach for hard-exploration problems". In: *arXiv preprint arXiv:1901.10995*.

Eysenbach, Benjamin, Russ R Salakhutdinov, and Sergey Levine (2019). "Search on the replay buffer: Bridging planning and reinforcement learning". In: *Advances in Neural Information Processing Systems* 32.

Forestier, Sebastien (2019). "Intrinsically Motivated Goal Exploration in Child Development and Artificial Intelligence: Learning and Development of Speech and Tool Use". PhD thesis. U. Bordeaux.

Fujimoto, Scott, David Meger, and Doina Precup (2019). "Off-policy deep reinforcement learning without exploration". In: *International conference on machine learning*. PMLR, pp. 2052–2062.

Goecks, Vinicius G et al. (2019). "Integrating behavior cloning and reinforcement learning for improved performance in dense and sparse reward environments". In: *arXiv preprint arXiv:1910.04281*.

Hansen, Steven et al. (2019). "Fast task inference with variational intrinsic successor features". In: *arXiv preprint arXiv:1906.05030*.

Justesen, Niels, Sebastian Risi, and Jean-Baptiste Mouret (2019). "Map-elites for noisy domains by adaptive sampling". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 121–122.

Kumar, Aviral et al. (2019). "Stabilizing off-policy q-learning via bootstrapping error reduction". In: *Advances in Neural Information Processing Systems* 32.

Levy, Andrew et al. (2019). "Learning Multi-Level Hierarchies with Hindsight". In: *ICLR*.

Matheron, Guillaume, Nicolas Perrin, and Olivier Sigaud (2019). "The problem with DDPG: understanding failures in deterministic environments with sparse rewards". In: *arXiv preprint arXiv:1911.11679*.

Nachum, Ofir et al. (2019). "Algaedice: Policy gradient from arbitrary experience". In: *arXiv preprint arXiv:1912.02074*.

Nasiriany, Soroush et al. (2019). "Planning with goal-conditioned policies". In: *Advances in Neural Information Processing Systems* 32.

Paine, Tom Le et al. (2019). "Making efficient use of demonstrations to solve hard exploration problems". In: *arXiv preprint arXiv:1909.01387*.

Peng, Xue Bin et al. (2019). "Advantage-weighted regression: Simple and scalable off-policy reinforcement learning". In: *arXiv preprint arXiv:1910.00177*.

Pong, Vitchyr H et al. (2019). "Skew-fit: State-covering self-supervised reinforcement learning". In: *arXiv preprint arXiv:1903.03698*.

Reddy, Siddharth, Anca D. Dragan, and Sergey Levine (2019). "Sqil: Imitation learning via reinforcement learning with sparse rewards". In: *arXiv preprint arXiv:1905.11108*.

Stanley, Kenneth O et al. (2019). "Designing neural networks through neuroevolution". In: *Nature Machine Intelligence* 1.1, pp. 24–35.

Tsitsimpelis, Ioannis et al. (2019). "A review of ground-based robotic systems for the characterization of nuclear environments". In: *Progress in Nuclear Energy* 111, pp. 109–124. ISSN: 0149-1970. DOI: https://doi.org/10.1016/j.pnucene.2018.10.023. URL: https://www.sciencedirect.com/science/article/pii/S0149197018302750.

Agarwal, Rishabh, Dale Schuurmans, and Mohammad Norouzi (2020). "An optimistic perspective on offline reinforcement learning". In: *International Conference on Machine Learning*. PMLR, pp. 104–114.

Alpaydin, Ethem (2020). *Introduction to machine learning*. MIT press.

Bodnar, Cristian, Ben Day, and Pietro Lió (2020). "Proximal distilled evolutionary reinforcement learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04, pp. 3283–3290.

Campos, Víctor et al. (2020). "Explore, discover and learn: Unsupervised discovery of state-covering skills". In: *International Conference on Machine Learning*. PMLR, pp. 1317–1327.

Cideron, Geoffrey et al. (2020). *QD-RL: Efficient Mixing of Quality and Diversity in Reinforcement Learning*. arXiv: 2006.08505 [cs.AI].

Colas, Cédric et al. (2020). "Scaling map-elites to deep neuroevolution". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 67–75.

Dadashi, Robert et al. (2020). "Primal wasserstein imitation learning". In: *arXiv preprint arXiv:2006.04678*.

Doncieux, Stephane et al. (2020). "Novelty Search Makes Evolvability Inevitable". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. GECCO '20. Cancún, Mexico: Association for Computing Machinery, pp. 85–93. ISBN: 9781450371285.

Flageat, Manon and Antoine Cully (2020). "Fast and stable MAP-Elites in noisy domains using deep grids". In: *arXiv preprint arXiv:2006.14253*.

Gaier, Adam, Alexander Asteroth, and Jean-Baptiste Mouret (2020). "Discovering representations for black-box optimization". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 103–111.

Jauhri, Snehal, Carlos Celemin, and Jens Kober (2020). "Interactive imitation learning in state-space". In: *arXiv preprint arXiv:2008.00524*.

Keller, Leon et al. (2020). "Model-based quality-diversity search for efficient robot learning". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 9675–9680.

Kuznetsov, Arsenii et al. (2020). "Controlling overestimation bias with truncated mixture of continuous distributional quantile critics". In: *International Conference on Machine Learning*. PMLR, pp. 5556–5566.

Matheron, Guillaume (2020). "Integrating motion planning into reinforcement learning to solve hard exploration problems". PhD thesis. Sorbonne Université.

Matheron, Guillaume, Nicolas Perrin, and Olivier Sigaud (2020). "PBCS: Efficient exploration and exploitation using a synergy between reinforcement learning and motion planning". In: *International Conference on Artificial Neural Networks*. Springer, pp. 295–307.

Paolo, Giuseppe et al. (2020). "Unsupervised learning and exploration of reachable outcome space". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2379–2385.

Pitis, Silviu et al. (2020). "Maximum entropy gain exploration for long horizon multi-goal reinforcement learning". In: *International Conference on Machine Learning*. PMLR, pp. 7750–7761.

Ravichandar, Harish et al. (2020). "recent advances in robot learning from demonstration". In: *Annual review of control, robotics, and autonomous systems* 3, pp. 297–330.

Tsounis, Vassilios et al. (2020). "Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning". In: *IEEE Robotics and Automation Letters* 5.2, pp. 3699–3706.

Yan, Mengyuan et al. (2020). "Learning topological motion primitives for knot planning". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 9457–9464.

Zhu, Henry et al. (2020). "The ingredients of real-world robotic reinforcement learning". In: *arXiv preprint arXiv:2004.12570*.

Amin, Susan et al. (2021). "A survey of exploration methods in reinforcement learning". In: *arXiv preprint arXiv:2109.00157*.

Arora, Saurabh and Prashant Doshi (2021). "A survey of inverse reinforcement learning: Challenges, methods and progress". In: *Artificial Intelligence* 297, p. 103500.

Bagaria, Akhil, Jason K Senthil, and George Konidaris (2021). "Skill discovery for exploration and planning using deep skill graphs". In: *International Conference on Machine Learning*. PMLR, pp. 521–531.

Bagaria, Akhil, Jason K Senthil, Matthew Slivinski, et al. (2021). "robustly Learning Composable Options in Deep Reinforcement Learning". In: *Proceedings of the 30th International Joint Conference on Artificial Intelligence*.

Chane-Sane, Elliot, Cordelia Schmid, and Ivan Laptev (2021). "Goal-conditioned reinforcement learning with imagined subgoals". In: *International Conference on Machine Learning*. PMLR, pp. 1430–1440.

Chenu, Alexandre et al. (2021). "Selection-Expansion: A Unifying Framework for Motion-Planning and Diversity Search Algorithms". In: *International Conference on Artificial Neural Networks*. Springer, pp. 568–579.

Colas, Cédric (June 2021). "Towards Vygotskian Autotelic Agents : Learning Skills with Goals, Language and Intrinsically Motivated Deep Reinforcement Learning". Theses. Université de Bordeaux. URL: https://theses.hal.science/tel-03337625.

Ecoffet, Adrien et al. (2021). "First return, then explore". In: *Nature* 590.7847, pp. 580–586.

Eysenbach, Benjamin and Sergey Levine (2021). "Maximum entropy RL (provably) solves some robust RL problems". In: *arXiv preprint arXiv:2103.06257*.

Fujimoto, Scott and Shixiang Shane Gu (2021). "A minimalist approach to offline reinforcement learning". In: *Advances in neural information processing systems* 34, pp. 20132–20145.

Gupta, Abhishek et al. (2021). "Reset-free reinforcement learning via multi-task learning: Learning dexterous manipulation behaviors without human intervention". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6664–6671.

Johns, Edward (2021). "Coarse-to-fine imitation learning: robot manipulation from a single demonstration". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4613–4619.

Kamienny, Pierre-Alexandre et al. (2021). "Direct then Diffuse: Incremental Unsupervised Skill Discovery for State Covering and Goal Reaching". In: *arXiv preprint arXiv:2110.14457*.

Kumar, Aviral et al. (2021). "Should I Run Offline Reinforcement Learning or Behavioral Cloning?" In: *Deep RL Workshop NeurIPS 2021*.

Martin, Jesus Bujalance, Raphaël Chekroun, and Fabien Moutarde (2021). "Learning from demonstrations with SACr2: Soft Actor-Critic with reward relabeling". In: *arXiv preprint arXiv:2110.14464*.

Nilsson, Olle and Antoine Cully (2021). "Policy Gradient Assisted MAP-Elites". In: *Genetic and Evolutionary Computation Conference*.

Raffin, Antonin et al. (2021). "Stable-Baselines3: reliable reinforcement Learning Implementations". In: *Journal of Machine Learning Research*.

Rakicevic, Nemanja, Antoine Cully, and Petar Kormushev (2021). "Policy Manifold Search: Exploring the Manifold Hypothesis for Diversity-based Neuroevolution". In: *Genetic and Evolutionary Computation Conference*.

Templier, Paul, Emmanuel Rachelson, and Dennis G Wilson (2021). "A geometric encoding for neural network evolution". In: *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 919–927.

Wu, Yuchen, Melissa Mozifian, and Florian Shkurti (2021). "Shaping rewards for reinforcement learning with imperfect demonstrations using generative models". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6628–6634.

Akakzia, Ahmed et al. (2022). "Help Me Explore: Minimal Social Interventions for Graph-Based Autotelic Agents". In: *arXiv preprint arXiv:2202.05129*.

Allard, Maxime et al. (2022). "Hierarchical Quality-Diversity for Online Damage Recovery". In: *arXiv preprint arXiv:2204.05726*.

Castanet, Nicolas, Sylvain Lamprier, and Olivier Sigaud (2022). "Stein Variational Goal Generation For Reinforcement Learning in Hard Exploration Problems". In: *arXiv preprint arXiv:2206.06719*.

Celemin, Carlos et al. (2022). "Interactive imitation learning in robotics: A survey". In: *Foundations and Trends® in Robotics* 10.1-2, pp. 1–197.

Chalumeau, Felix et al. (2022). "Assessing Quality-Diversity Neuro-Evolution Algorithms Performance in Hard Exploration Problems". In: *arXiv preprint arXiv:2211.13742*.

Chenu, Alexandre, Nicolas Perrin-Gilbert, and Olivier Sigaud (2022). "Divide & Conquer Imitation Learning". In: *arXiv preprint arXiv:2204.07404*.

Chenu, Alexandre, Olivier Serris, et al. (2022). "Leveraging Sequentiality in Reinforcement Learning from a Single Demonstration". In: *arXiv preprint arXiv:2211.04786*.

Colas, Cédric et al. (2022). "Autotelic agents with intrinsically motivated goal-conditioned reinforcement learning: a short survey". In: *Journal of Artificial Intelligence Research* 74, pp. 1159–1199.

Contributors, MuJoCo Menagerie (2022). *MuJoCo Menagerie: A collection of high-quality simulation models for MuJoCo*. URL: http://github.com/deepmind/mujoco_menagerie.

Gallouédec, Quentin and Emmanuel Dellandréa (2022). "Cell-Free Latent Go-Explore". In: *arXiv preprint arXiv:2208.14928*.

Gupta, Sthithpragya, Aradhana Nayak, and Aude Billard (2022). "Learning High Dimensional Demonstrations Using Laplacian Eigenmaps". In: *arXiv preprint arXiv:2207.08714*.

Huber, Johann et al. (2022). "E2r: a Hierarchical-Learning inspired Novelty-Search method to generate diverse repertoires of grasping trajectories". In: *arXiv preprint arXiv:2210.07887*.

Hutsebaut-Buysse, Matthias, Kevin Mets, and Steven Latré (2022). "Hierarchical Reinforcement Learning: A Survey and Open Research Challenges". In: *Machine Learning and Knowledge Extraction* 4.1, pp. 172–221.

Lim, Bryan et al. (2022). "Dynamics-aware quality-diversity for efficient learning of skill repertoires". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5360–5366.

Mészáros, Anna, Giovanni Franzese, and Jens Kober (2022). "Learning to Pick at Non-Zero-Velocity From Interactive Demonstrations". In: *IEEE Robotics and Automation Letters* 7.3, pp. 6052–6059.

Morel, Aurélien et al. (2022). "Automatic Acquisition of a Repertoire of Diverse Grasping Trajectories through Behavior Shaping and Novelty Search". In: *arXiv preprint arXiv:2205.08189*.

Perrin-Gilbert, Nicolas (2022). *xpag: a modular reinforcement learning library with JAX agents*. URL: https://github.com/perrin-isir/xpag.

Pierrot, Thomas et al. (2022). "Diversity Policy Gradient for Sample Efficient Quality-Diversity Optimization". In: *ICLR Workshop on Agent Learning in Open-Endedness*.

Prudencio, Rafael Figueiredo, Marcos Roa Maximo, and Esther Luna Colombini (2022). "A Survey on Offline reinforcement Learning: Taxonomy, Review, and Open Problems". In: *arXiv preprint arXiv:2203.01387*.

Sigaud, Olivier (2022). "Combining Evolution and Deep Reinforcement Learning for Policy Search: a Survey". In: *arXiv preprint arXiv:2203.14009*.

Tranzatto, Marco et al. (2022). "Cerberus: Autonomous legged and aerial robotic exploration in the tunnel and urban circuits of the darpa subterranean challenge". In: *arXiv preprint arXiv:2201.07067*.

Wu, Philipp et al. (2022). "Daydreamer: World models for physical robot learning". In: *arXiv preprint arXiv:2206.14176*.