Divide & Conquer Imitation Learning

Alexandre Chenu, Nicolas Perrin-Gilbert, Olivier Sigaud

Divide & Conquer Imitation Learning

Alexandre Chenu[1], Nicolas Perrin-Gilbert[1] and Olivier Sigaud[1]

***Abstract*— When cast into the Deep Reinforcement Learning framework, many robotics tasks require solving a long horizon and sparse reward problem, where learning algorithms struggle. In such context, Imitation Learning (IL) can be a powerful approach to bootstrap the learning process. However, most IL methods require several expert demonstrations which can be prohibitively difficult to acquire. Only a handful of IL algorithms have shown efficiency in the context of an extreme low expert data regime where a single expert demonstration is available. In this paper, we present a novel algorithm designed to imitate complex robotic tasks from the states of an expert trajectory. Based on a sequential inductive bias, our method divides the complex task into smaller skills. The skills are learned into a goal-conditioned policy that is able to solve each skill individually and chain skills to solve the entire task. We show that our method imitates a non-holonomic navigation task and scales to a complex simulated robotic manipulation task with very high sample efficiency.**

I. Introduction

Deep Reinforcement Learning (DRL) has been successful in solving complex simulated ([1], [2], [3]) and physical robotic control problems [4]. However, even in simulation, DRL is still limited when applied to complex tasks including sparse reward signals [5], long control-time horizons and critical states [6]. In this context, Imitation Learning (IL) is a fruitful alternative to failing DRL algorithms. In IL, a number of expert demonstrations are used to guide the learning process so that the behavior of an agent matches that of an expert.

However, most imitation learning algorithms require a large set of expert demonstrations which can be hard to acquire, particularly in the context of long-horizon problems. In this context, a few methods strive to design an IL algorithm that can work with a single demonstration. Among these methods, the Go-explore approach ([7], [3] relies on a strategy called Backplay [8], [9]) which learns a single controller by starting further and further away from the final point. As it needs to learn and play many longer and longer trajectories, this approach suffers from sample inefficiency. Another recent approach that can handle single demonstrations is PWIL [10], which uses offline learning to define an episodic reward function based on the demonstration, and performs IL without formulating it as an adversarial learning problem, contrary to most recent approaches. This improves the efficiency and stability of the IL process, but this still resorts to performing DRL on a long horizon task, which limits the potential gain in sample efficiency.

In this paper, we present Divide & Conquer Imitation Learning (DCIL), a DRL-based IL algorithm relying on a sequential inductive bias to solve long-horizon imitation tasks using a single demonstration. As the name implies, DCIL divides the complex task into skills. The skills are learned into a goal-conditioned policy (GCP) that is able to solve each skill individually and chain skills to solve the entire task. In complex problems, it may be necessary for the goal space to have a lower dimensionality than the state space, which means that skills may lead to states that do not match the expert demonstration. As a result, the chaining of the skills becomes challenging, and we address this issue by introducing, for each skill, a chaining reward bonus that depends on a value function learned over the next skill. We first evaluate our approach in a toy Dubins maze environment where the dynamics of the controlled system is constrained, and show that our chaining mechanism plays a crucial role in ensuring the success of the method, resulting in a sample efficiency that is several orders of magnitude better than that of Backplay and PWIL. 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,

Fig. 1: Chaining of four skills learned using DCIL in a simulated object 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 or not.

[1]Sorbonne Université, CNRS, Institut des Systèmes Intelligents et de Robotique, ISIR F-75005 Paris, France chenu@isir.upmc.fr

and demonstrate an even greater gain in sample efficiency compared to Backplay, the method used by Go-Explore on this benchmark.

II. Related work

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 [11], which relies on regression to learn a policy that mimics the actions of the expert. A more indirect approach is Inverse Reinforcement Learning (IRL) [12], 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 actions of the expert, and by their typical need for many demonstrations. More specifically, with few demonstrations, BC tends to suffer from compounding error caused by covariate shift [13]. In the case of IRL, it can be difficult to extract a reward from a unique demonstration. For instance, popular adversarial methods for IRL ([14], [15], [16], [17]) rely on a generator-discriminator architecture that may become unstable if the discriminator is not trained on sufficiently many samples from expert trajectories.

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

A. IL from a single demonstration

1) PWIL: Primal Wasserstein Imitation Learning (PWIL) [10] is a recent IRL method that minimizes a greedy version of the Wasserstein distance between the state-action distributions of the agent and the expert. The Wasserstein distance presents several good properties that have been proficiently used in the Deep Learning community [18]. PWIL solves an occupancy matching problem between an agent and the demonstration without relying on adversarial training, which makes it more stable than adversarial methods. More 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 one single demonstration.

2) Backplay: The Backplay algorithm ([8], [9]), 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 [7] 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 V-D. In Backplay, the objective is to reach the final state of the expert demonstration. The RL agent is initialized close to the rewarding state and the starting state is progressively moved backward along the demonstration if it is successful enough at reaching the desired state. Backplay can be seen as a curriculum for RL approaches in the context of sparse reward and long-horizon control [19].

B. Skill-chaining

In this paper, we propose to address the single demonstration imitation problem by transforming a demonstration into a sequence of RL tasks. This divide & conquer type of strategy is a common way to solve a complex RL problem by learning a set of policies on simpler tasks and chaining them to solve the global task [20]. For example, this principle is applied by the Backplay-Chain-Skill part of the Play-Backplay-Chain-Skill (PBCS) algorithm [21]. The Backplay algorithm is used to learn a set of skills backward from the final state of a single demonstration obtained using a planning algorithm. However, in PBCS, the agent must reach the neighborhood of a precise state to transit from one skill to the next. In high-dimensional states, the constraint of reaching a sequence of precise states quickly becomes a very hard learning problem, and as a result PBCS struggles to scale to complex robotic tasks.

There are similar approaches in the recent skill-chaining literature ([22], [23]), in which skills are formalized using the option framework ([24], [25]). An option is composed of an initial set of states in which this option can be activated, a termination function which 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) [22], for a given goal, a first option is learned to reliably reach it from a nearby region. 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. This framework has not been applied to imitation, but it is possible that a modified version could address IL. However, with a single demonstration and a complex problem, the initiation conditions could end up being very small and precise, in a similar way to PBCS, with the same difficulty to scale to high-dimensional problems. Some adversarial approaches rely on the framework of options to efficiently perform IL (e.g. [17]), but as other AIL-based methods, they tend to fail in the low expert data regime that we consider.

In our method, we consider a goal space as a low-dimensional projection of the state space. Instead of targeting the neighborhood of a precise state to complete a skill as in PBCS, we aim at the neighborhood of a low-dimensional goal. Moreover, skills are not performed by independent policies as in the option framework. Instead, we learn a single goal-conditioned policy able to perform different skills depending on the goal it is conditioned on. Finally, skills are not trained independently. Along the chain of skill, the policy is trained in order to complete a skill by reaching states that are compatible with the execution of the following skill.

III. Background

In our proposed approach, we extract a sequence of targets from the demonstration, and rely on the formalism of Goal-Conditioned Reinforcement Learning (GCRL) to learn a unique policy able to reach the consecutive targets in order.

A. Goal-conditioned Reinforcement Learning

A DRL problem is described by a state space $\mathscr{S}$, an action space $\mathscr{A}$, an unknown reward function $R : \mathscr{S} \times \mathscr{A} \to \mathbb{R}$, an unknown transition probability $p(s_{t+1}|s_t, a_t)$, and potentially a distribution of initial states. In a finite horizon setting, an episode has a maximum length of T_{max} control steps. A GCRL problem extends the RL formalism to a multiple goal setting where the reward function $R : \mathscr{S} \times \mathscr{A} \times \mathscr{G} \to \mathbb{R}$ depends on the goal that is considered. In addition to sampling an initial state, a goal is sampled at the beginning of each episode using a goal distribution $\rho_{\mathscr{G}}$.

The objective in GCRL is to obtain a goal-conditioned policy (GCP) [26] $\pi(a|s, g)$ that maximizes the expected cumulative rewards $\mathbb{E}[\sum_{t=0}^{T_{max}} R(s_t, g)]$ for a goal g.

B. Distance-based sparse reward

In most DCRL settings, the reward signal is sparse as the goal-conditioned agent only receives a reward for achieving the desired goal g.

In our method, we consider a common version of GCRL where goals represent low dimensional projections of states. A state is projected to a goal according to a mapping $p_{\mathscr{G}} : \mathscr{S} \to \mathscr{G}$ associated to the definition of the goal space. To achieve a goal, the agent must transit to any state $s \in \mathscr{S}$ that can be mapped to a goal $g_s = p_{\mathscr{G}}(s) \in \mathscr{G}$ within a distance less than $\epsilon_{success}$ from g, $\epsilon_{success}$ being an environment-dependent hyper-parameter. Those *success states* form the success state set $\mathscr{S}_g$ associated with goal g and their corresponding goals constitute its success goal set $\mathscr{G}_g$. We use the common L2-norm to compute the distance between two goals but other norms can be considered. Note that reaching a goal corresponding to a low-dimensional projection of a state does not fully condition the state that the agent is in. This can be very problematic when chaining two skills as illustrated in Figure 2 and discussed in Section IV-C.3.

The environment-agnostic reward function is defined as:

$$R(s, g) = \begin{cases} 1 \text{ if } s \in \mathscr{S}_g \\ 0 \text{ otherwise.} \end{cases} \quad (1)$$

To assess how much reward can be expected by following policy π conditioned on a goal g from state s, we use a goal-conditioned value function V^π defined as the expected sum of future rewards, given s and g:

$$V^\pi(s, g) = \mathbb{E}[\sum_{t=0}^{T_{max}} R(s_t, g)|s, \pi(., g)]. \quad (2)$$

This value function is the central tool used to compute the chaining reward bonus R_{bonus} (see Section IV-C.3).

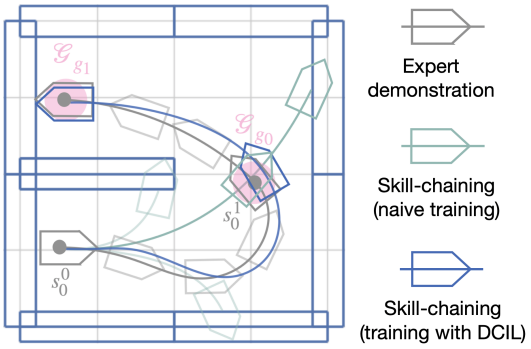

Fig. 2: In this example, an expert demonstrates how to navigate a Dubins car (grey trajectory) in a simple 2D maze. The demonstration is split into a set of skills. Here, the two skills consists in reaching g_0 and g_1, two x-y positions represented as $\mathscr{G}_{g_0}$ and $\mathscr{G}_{g_1}$, whereas states are three-dimensional and include the orientation of the car. If the GCP is trained naively, the agent could solve the first skill by reaching states with an invalid orientation for the next skill (green trajectory). DCIL helps the agent to complete skills by reaching success states with a valid orientation to successfully chain the skills (blue trajectory).

C. Relabelling

Exploring the state space in the context of sparse reward can be challenging even for modern deep RL algorithms ([27], [28], [29]). To simplify exploration in the GGI framework, GCRL agents often use the Hindsight Experience Replay (HER) relabelling technique [1]. If the agent fails to reach the goal it is conditioned on, HER relabels the transitions of the episode by replacing the goal initially intended with the goal it accidentally achieved.

IV. Methods

The Divide & Conquer Imitation Learning (DCIL) algorithm is designed to solve what can be called 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 learning a sequence of goal-conditioned chainable skills. 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, especially in a robotic context, and as we show in Section V, the GGI approach can significantly accelerate the IL process.

A. Goal-Guided Imitation

The Goal-Guided Imitation (GGI) framework can be formulated as a variant of GCRL. A set of N_{skills} goal-based skills are extracted from a single expert demonstration $\tau_e = \{s_0^e, s_1^e, ..., s_N^e\}$ and the objective is to obtain a GCP that is able to complete each skill sequentially. From this trajectory τ_e, we derive skills by extracting a set of goals $(g_i)_{i\in[0,N_{skills}]} \in \mathscr{G}^{N_{skills}}$. Each skill K_i is defined by its goal g_i. To avoid any ambiguity, we call skill-goals the

goals g_i associated with skill K_i. The objective in the GGI framework is to obtain a GCP that is able to reach skill-goals $g_{i+1}, g_{i+2}, ...$ after completing skill K_i. This GCP is then used to chain the successive skills in order to reach the final state s_N^e of the expert trajectory. Unlike in GCRL, in the GGI framework, ρ_g is a distribution of skill-goals only, as only skill-goals may be sampled to condition the GCP.

B. DCIL hypotheses

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

1) Reset: Training the GCP in DCIL (see Section IV-C.2) 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 forms of reset such as the assumption that the agent can be reset in uniformly sampled states (*reset-anywhere*) have also been considered in the GCRL literature [30]. PWIL is based on the more classical assumption of a unique reset, and BC does not require any reset at all.

2) No expert actions: Similarly to Backplay, the imitation in DCIL is solely based on the expert trajectory in the state space. Learning from states only is crucial when the expert actions are difficult to collect (e.g. human demonstrations). On the contrary, both PWIL and BC require state-action demonstrations.

3) Goal-space definition: As a GCRL-based method, DCIL 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 uses a GCP.

C. The Divide & Conquer Imitation Learning algorithm

In DCIL, we extract skills goals and initial states from the expert trajectory (Section IV-C.1). The GCP is then trained to perform each skill using a DRL algorithm. Training for a skill boils down to starting in the associated initial state and completing a local rollout to reach the skill-goal (Section IV-C.2). While the agent learns a skill, it is encouraged to complete it by reaching states that are compatible with the execution of the next ones (Section IV-C.3). Finally, the agent can recover the expert behavior by chaining the skills sequentially (Section IV-C.4). These different stages of DCIL are detailed in the four next sections and summarized in Algorithms 1 and 2.

1) Extracting skills from the expert trajectory: To transform the expert trajectory into teachable skills, we project it in the goal space and divide it into N_{skill} sub-trajectories $(\tau_i)_{i\in[0,N_{skill}]}$ of equal arc lengths ϵ_{dist}. For each sub-trajectory, we extract one tuple (s_0^i, g_i, T_{max}^i) that we associate to a skill, where s_0^i corresponds to the demonstrated state that resulted in the initial goal of the sub-trajectory, g_i the initial goal of the next sub-trajectory and $T_{max}^i = \beta|\tau_i|$, where $|\tau_i|$ is the length of the sub-trajectory in time steps, and $\beta > 1$ is a predefined coefficient[1] used to facilitate exploration while learning the skill (see Section IV-C.2). The initial goal of the next sub-trajectory g_i constitutes the skill-goal. The initial state s_0^i and the length T_{max}^i are used to learn the skill.

Algorithm 1 DCIL - GCP training

Input: $\pi_\phi, Q_\theta, Q_{\bar\theta}$ ▷ actor, critic and target critic networks
$B \leftarrow []$ ▷ replay-buffer
for $n = 1 : N_{episode}$ **do**
  $(s_0^n, T_{max}^n, g_n) \leftarrow select\_skill()$ ▷ **Step 1**
  $s_t \leftarrow$ env.reset(s_0^n) ▷ **Step 2**
  $t \leftarrow 0$
  while not done **do**
    $a_t \leftarrow \pi_\phi(s_t|g_n)$
    $s_{t+1} \leftarrow env.step(a_t)$
    $r_t \leftarrow 0$
    if $|p_{\mathscr{G}}(s_{t+1}) - g_n|_2 \leq \epsilon_{success}$ **then**
      $success, done \leftarrow True, True$
    else
      $success, done \leftarrow False, False$
    if $t \geq T_{max}^n$ **then**
      $done \leftarrow True$
    $B \leftarrow B + [(s_t, a_t, s_{t+1}, r_t, g_n, done, success)]$
    $t \leftarrow t + 1$
    if success **then** ▷ **Step 3**
      $success, done \leftarrow False, False$
      $(\_, T_{max}^n, g_n) \leftarrow$ next_skill(g_n) ▷ overshoot
      $t \leftarrow 0$
    SAC_update$(\pi_\phi, Q_\theta, Q_{\bar\theta}, B)$ ▷ Algo 2

2) Learning the skills: To train the GCP on the different skills, DCIL runs a three-step loop.

a) Step 1: DCIL selects a skill $K_i = (s_0^i, g_i, T_{max}^i)$ to train on (function select_skill in Algorithm 1) and resets the environment in s_0^i. Note that the selection of skills is biased towards skills with a low ratio of successful rollouts over the total number of trials for these skills. We implemented such distribution using a fitness proportionate selection [31] where the fitness corresponds to the inverse of this ratio.

b) Step 2: DCIL conditions the GCP on g_i and the agent performs a rollout to complete the skill which is interrupted either if the agent reaches $\mathscr{S}_{g_i}$ or if skill length $T^i{}_{max}$ is exceeded.

c) Step 3: When the agent successfully completes skill K_i, DCIL applies an overshoot mechanism (see Section IV-C.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 [32] of the critic and actor networks including the chaining reward bonus (see Section IV-C.3). In a sampled batch of transitions, half of the

[1] In our experiments, we use β = 1.25.

Algorithm 2 modified SAC update (+ HER + Chaining Reward Bonus)

Input: $\pi_\phi, Q_\theta, Q_{\bar{\theta}}, B$ ▷ actor, critic and target critic networks
$batch_{HER} \leftarrow HER(B)$ ▷ HER relabelling
$batch \leftarrow B$ ▷ no HER relabelling
for $(s_t^k, a_t^k, s_{t+1}^k, g^k, r^k, done, success)$ in $batch$ **do**
  if success **then** ▷ Chaining reward bonus
    $(\_, \_, g^{k'}) \leftarrow$ next_skill(g^k)
    $r^k \leftarrow 1 + Q_{\bar{\theta}}(s_{t+1}^k, \pi_\phi(s_{t+1}^k, g^{k'}), g^{k'})$
$batch \leftarrow batch + batch_{HER}$
for $gradient\_step = 1 : N_{gradient\_step}$ **do** ▷ see [32]
  $\pi_\phi, Q_\theta, Q_{\bar{\theta}} \leftarrow gradient\_step(\pi_\phi, Q_\theta, Q_{\bar{\theta}}, batch)$
Output: $\pi_\phi, Q_\theta, Q_{\bar{\theta}}$

transitions are relabelled using HER. This three-step loop is summarized in Algorithm 1.

3) Ensuring successful skill chaining: To ensure that skills can be chained, we use an *overshoot mechanism* and a *chaining reward bonus*.

a) Overshoot: Successfully chaining the skills requires that the agent completes any first skill by reaching states from which it is able to perform the next ones. When the agent completes a skill, it can reach either *valid initial states* in $\mathscr{I}_i^{valid}$ from which it will successfully perform the next skills or *invalid initial states* from which it will not (see Figure 3). During a training rollout for a skill, if the agent reaches a success state, the overshoot mechanism immediately conditions the GCP on the skill-goal of the next skill (function next_skill in Algorithm 1). The agent instantly starts a new rollout for this next skill from its current state.

With these overshoot rollouts, the agent can learn how to perform the next skills while starting from other initial states than the ones extracted from the demonstration. As the agent progresses at performing skills from those different initial states, some previously invalid initial states become valid. So the purpose of the overshoot mechanism is to make $\mathscr{I}_{valid}$ grow. However, in complex environments (e.g. under-actuated and non-holonomic environments) not all invalid initial states can become valid. Therefore, another mechanism is necessary to help the agent to complete skills by only reaching valid starting states.

b) Chaining reward bonus: To facilitate skill completion by only reaching valid initial states for the next skills, we add a chaining reward bonus to the sparse reward received by the agent each time it successfully completes a skill. The chaining reward bonus is defined as the goal-conditioned value function $V^\pi(., g_{i+1})$ of the agent, conditioned on the skill-goal of the next skill (see Algorithm 2 for additional information on goal-conditioned value computation). Therefore, the modified reward function is defined as:

$$\bar{R}(s, g_i) = \begin{cases} 1 + V^\pi(s, g_{i+1}) \text{ if } s \in \mathscr{S}_{g_i} \\ 0 \text{ otherwise.} \end{cases} \quad (3)$$

The idea behind this bonus is that valid initial 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 initial states, the rewards from skills $K_{i+1}, K_{i+2}, ...$ are propagated backward up to valid initial states.

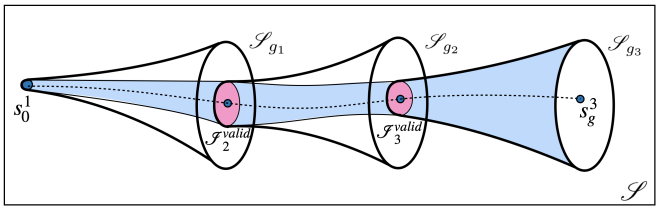

Fig. 3: To retrieve the expert behavior (dotted line), the agent has to perform each skill (represented as funnels) sequentially. To successfully chain the skills the agent must transit between their set of valid initial states $\mathscr{I}_i^{valid}$ (pink disks) via the blue-shaded path and avoid solving a skill by reaching the other success states in $\mathscr{S}_{g_i} \setminus \mathscr{I}_i^{valid}$.

In complex environments, if $\mathscr{I}_i^{valid}$ only contains a few isolated valid initial 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 or an adaptive one, forces the agent to explore diverse trajectories to reach $\mathscr{S}_{g_i}$ [33]. It helps the agent to eventually find a path towards valid initial state and propagate the chaining bonus reward.

4) 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 skill-goal g_0 of the first skill K_0. After completing this first skill, the agent is conditioned on g_1 and solves K_1. This process is repeated for every skill K_i until the final skill is completed.

V. Experiments

In this section, we introduce the experimental setup used to evaluate DCIL (Section V-A), we present an ablation study of the two main components of DCIL (Section V-B) and we compare DCIL to three baselines: BC, Backplay and PWIL (Section V-C). The code of DCIL based on Stable Baselines 3 [34] is provided here: https://github.com/AlexandreChenu/dcil.

A. Experimental setup

We evaluate DCIL in two environments: the *Dubins Maze* environment that we introduce and the *Fetch* environment presented in [3].

1) Dubins Maze: In the Dubins Maze environment, the agent controls a Dubins car [35] 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 skill-goal conditioning the GCP, the agent can easily reach a success state for a given skill with an orientation that is invalid for the next skills. Demonstrations for this environment are obtained using the Rapidly-Exploring Random Trees algorithm [36].

2) Fetch: We also evaluate DCIL in the simulated grasping task for a 8 degrees-of-freedom deterministic robot manipulator. This environment was presented in the First Return then Explore paper [3]. 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 which contains the Cartesian and angular positions and the velocity 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 the 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 [7].

3) Baselines: We compare DCIL 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 ([9], [8]) and PWIL [10]. A comparison of the different key assumptions required by each algorithm is detailed in Section IV-B. 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 [3].

B. Ablation study

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

The performance shown in Figure 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 using the chaining reward bonus (DCIL full and DCIL w/o overshoot) outperform the other two variants. Besides, only the full version of DCIL recovers the expert behavior in 100% of trials. Finally, DCIL also benefits from the overshoot mechanism as its success rate increases faster than DCIL without overshoot during the 50.000 first training steps.

Figure 5 presents a training run of the full version of DCIL. Although the GCP is training on skills separately, it is able to chain them in order to recover the expert behavior and to navigate the maze. In order to visualize how the chaining reward bonus encourages the agent to complete the skills by reaching valid initial states, we evaluated DCIL and DCIL w/o chaining bonus reward in a simplified version of the

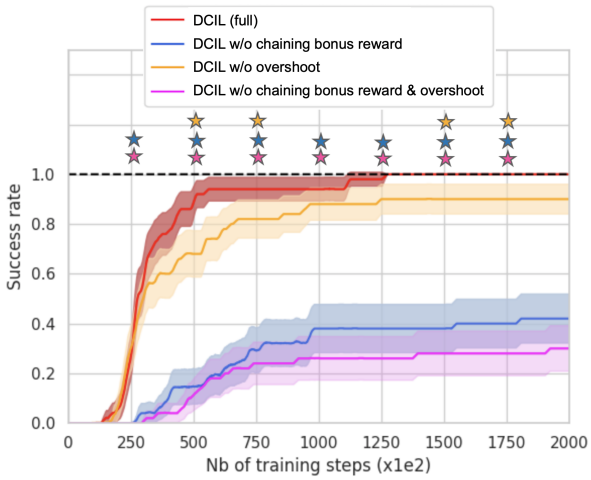

Fig. 4: Ablation study of DCIL in the Dubins Maze environment. We evaluate the success rate of four versions of DCIL (full DCIL DCIL w/o overshoot, DCIL w/o chaining bonus reward, DCIL w/o both) throughout training. Means and standard deviations ranges over 30 total runs (3 random seeds for 10 different expert trajectories). Stars indicate significant differences over DCIL (full) as reported by Welch's t-test with $\alpha = 0.05$ [37].

Dubins Maze where the agent only has two skills to chain. As Figure 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.

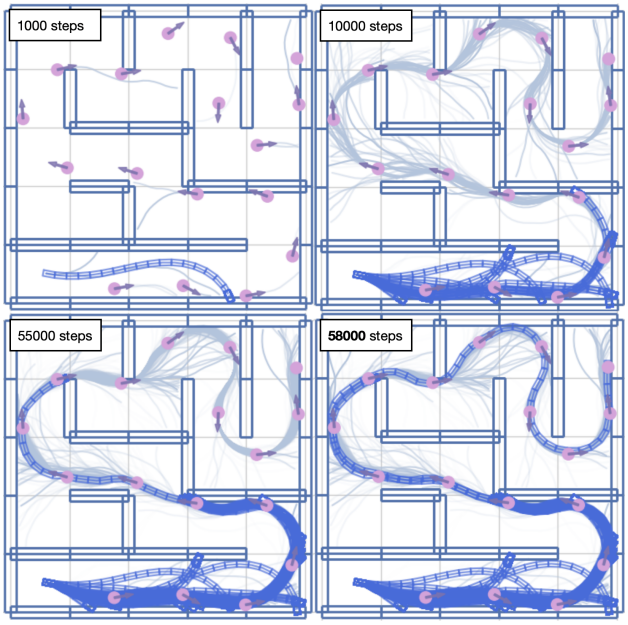

Fig. 5: Training run of DCIL in the Dubins Maze. For each skill, the initial state is represented by a purple arrow and the success goal set by a pink disk. Grey lines correspond to skills training rollouts. Blue car trajectories correspond to the agent skill chaining every 1000 training steps.

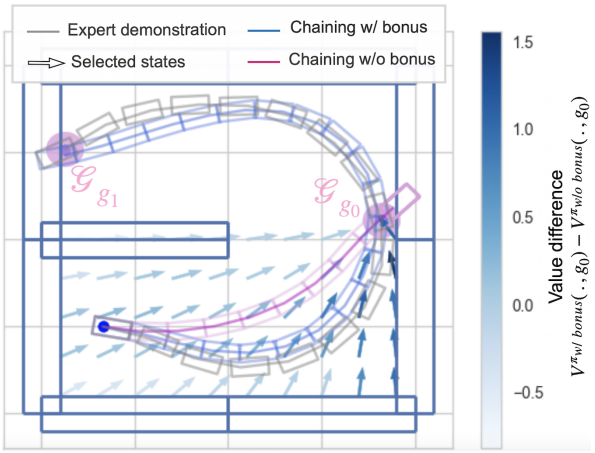

Fig. 6: Learning the first skill (reaching $\mathscr{G}_{g_0}$) without chaining reward bonus prevents the agent from completing it by reaching an valid initial state for the second skill (reaching $\mathscr{G}_{g_1}$) as illustrated by the purple trajectory. The chaining reward bonus increases the value 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 ($V^{\pi_{w/o\ bonus}}(.,g_0)$).

C. Comparison to baselines in Dubins Maze

Figure 7 evaluates how DCIL 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 use a metric based on 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 is the only method able 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 trains on very short rollouts compare to PWIL which trains on fixed-length episodes and Backplay which trains on episodes of increasing length.

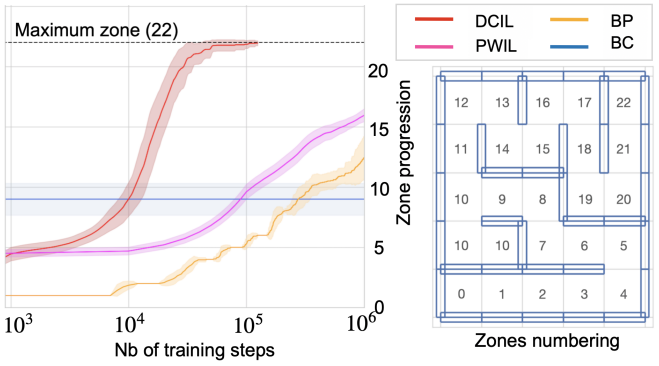

Fig. 7: Comparison of DCIL to Backplay, PWIL and BC with a single demonstration. We evaluate the progression through the Dubins Maze Environment (left) using 22 zones (right). For DCIL, PWIL and BC, the progression corresponds to the maximum zone reached during evaluation. For Backplay, the progression corresponds to the difference between the highest zone number and the zone from which the agent started. Means and standard deviations ranges over 3 random seeds for 10 different expert trajectories (30 total runs for each variant) for each method.

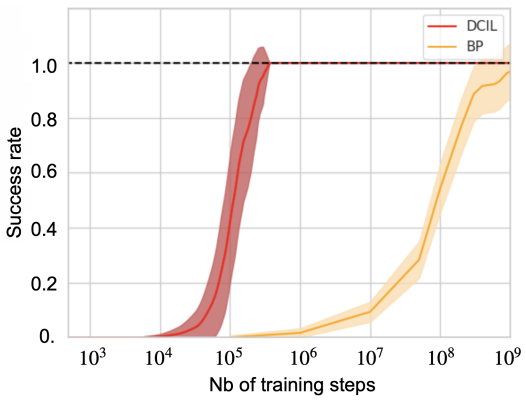

Fig. 8: Comparison of DCIL to Backplay in the Fetch environment. The results of DCIL present the mean and standard deviation over 5 seeds for 5 expert demonstrations (25 total runs). The results of Backplay are extracted from [3] and could not be reproduced despite running the author's code with the same hyper-parameters. For DCIL, in $\sim 10\%$ of the runs, the critic networks used in SAC diverge which results in failed runs. Only the runs that did not diverge are presented here.

D. Scaling to a complex object manipulation task

While the experiments in the Dubins Maze demonstrate the performance of DCIL 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 a much more complicated dynamic.

Figure 8 evaluates how DCIL performs when trained on $1e6$ training interactions. We compare our approach to the performance of Backplay presented in [3]. As in the Dubins Maze, DCIL solves the Fetch task three orders of magnitude faster than Backplay. DCIL is able to learn the full fetch behavior by training only on short rollouts.

Discussion & Conclusion

In this paper, we have introduced Divide & Conquer Imitation Learning (DCIL), an imitation learning algorithm solving long-horizon tasks using a single demonstration. DCIL relies on a sequential inductive bias and adopts a divide & conquer strategy to learn smaller skills that, chained together, solve the long-horizon task. In order for a goal-conditioned policy to learn each skill individually and to apply skill-chaining to recover the expert behavior, we introduced an *overshoot mechanism* and a *chaining reward bonus* that indirectly make skills aware of the next ones, and significantly improves the chainability of the skills. We

highlighted the key contribution of both mechanisms in the performance of DCIL by conducting an ablation study in a maze environment with a Dubins car. Moreover, we showed the efficiency of DCIL 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, in future work, will be critical when applying the method to physical robots. Yet, the application of DCIL to physical robots would require at least one modification of the algorithm which concerns the reset assumption. The usual option would be to replace resets to any previously encountered state by a unique fixed reset. This could be done by learning a way to "return to interesting states", an approach that has been studied in one of the variants of the Go-Explore algorithm [3]. However, even this assumption of a single reset can be troublesome with physical robots, especially when object manipulation is involved. For this reason, our main research direction will be to consider the possibility of learning small robotic skills that are not only chainable but *reversible*, which in particular robotic contexts could lead to a sample-efficient divide & conquer approach for imitation learning that would not require any kind of reset at all.

REFERENCES

[1] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight Experience Replay," *arXiv preprint arXiv:1707.01495*, 2017.

[2] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[3] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "First return, then explore," *Nature*, vol. 590, no. 7847, pp. 580–586, 2021.

[4] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, S. Levine, and G. Brain, "Time-contrastive networks: Self-supervised learning from video," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 1134–1141.

[5] G. Matheron, N. Perrin-Gilbert, and O. Sigaud, "The problem with DDPG: understanding failures in deterministic environments with sparse rewards," *arXiv preprint arXiv:1911.11679*, 2019.

[6] A. Kumar, J. Hong, A. Singh, and S. Levine, "Should i run offline reinforcement learning or behavioral cloning?" in *Deep RL Workshop NeurIPS 2021*, 2021.

[7] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "Go-explore: a new approach for hard-exploration problems," *arXiv preprint arXiv:1901.10995*, 2019.

[8] C. Resnick, R. Raileanu, S. Kapoor, A. Peysakhovich, K. Cho, and J. Bruna, "Backplay: "man muss immer umkehren"," *CoRR*, vol. abs/1807.06919, 2018.

[9] T. Salimans and R. Chen, "Learning Montezuma's Revenge from a Single Demonstration," *CoRR*, vol. abs/1812.03381, 2018.

[10] R. Dadashi, L. Hussenot, M. Geist, and O. Pietquin, "Primal Wasserstein Imitation Learning," *arXiv preprint arXiv:2006.04678*, 2020.

[11] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural computation*, vol. 3, no. 1, pp. 88–97, 1991.

[12] S. Russell, "Learning agents for uncertain environments," in *Proceedings of the eleventh annual conference on Computational learning theory*, 1998, pp. 101–103.

[13] S. Ross and D. Bagnell, "Efficient Reductions for Imitation learning," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 661–668.

[14] J. Ho and S. Ermon, "Generative adversarial imitation learning," *CoRR*, vol. abs/1606.03476, 2016.

[15] I. Kostrikov, K. K. Agrawal, S. Levine, and J. Tompson, "Addressing sample inefficiency and reward bias in inverse reinforcement learning," *CoRR*, vol. abs/1809.02925, 2018.

[16] Y. Ding, C. Florensa, M. Phielipp, and P. Abbeel, "Goal-conditioned imitation learning," *CoRR*, vol. abs/1906.05838, 2019.

[17] P. Henderson, W.-D. Chang, P.-L. Bacon, D. Meger, J. Pineau, and D. Precup, "OptionGAN: Learning joint reward-policy options using generative adversarial inverse reinforcement learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[18] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*. PMLR, 2017, pp. 214–223.

[19] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, "Reverse curriculum generation for reinforcement learning," in *Conference on robot learning*. PMLR, 2017, pp. 482–495.

[20] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, "Constructing Skill trees for Reinforcement Learning agents from demonstration trajectories," in *Advances in Neural Information Processing Systems*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds., vol. 23. Curran Associates, Inc., 2010.

[21] G. Matheron, N. Perrin-Gilbert, and O. Sigaud, "PBCS: Efficient exploration and exploitation using a synergy between reinforcement learning and motion planning," in *International Conference on Artificial Neural Networks*. Springer, 2020, pp. 295–307.

[22] A. Bagaria and G. Konidaris, "Option discovery using deep skill chaining," in *International Conference on Learning Representations*, 2019.

[23] A. Bagaria, J. Senthil, M. Slivinski, and G. Konidaris, "Robustly learning composable options in deep reinforcement learning," in *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, 2021.

[24] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[25] D. Precup, *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.

[26] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *International conference on machine learning*. PMLR, 2015, pp. 1312–1320.

[27] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "Vime: Variational information maximizing exploration," *Advances in neural information processing systems*, vol. 29, 2016.

[28] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," *Advances in neural information processing systems*, vol. 29, 2016.

[29] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2018.

[30] S. Nasiriany, V. Pong, S. Lin, and S. Levine, "Planning with goal-conditioned policies," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[31] T. Blickle and L. Thiele, "A Comparison of Selection Schemes Used in Evolutionary Algorithms," *Evolutionary Computation*, vol. 4, no. 4, pp. 361–394, 12 1996.

[32] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[33] B. Eysenbach and S. Levine, "Maximum entropy RL (provably) solves some robust RL problems," *arXiv preprint arXiv:2103.06257*, 2021.

[34] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning implementations," *Journal of Machine Learning Research*, 2021.

[35] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[36] S. M. LaValle *et al.*, "Rapidly-exploring Random Trees: A new tool for path planning," 1998.

[37] C. Colas, O. Sigaud, and P.-Y. Oudeyer, "A Hitchhiker's guide to statistical comparisons of reinforcement learning algorithms," *arXiv preprint arXiv:1904.06979*, 2019.