Region Growing Curriculum Generation for Reinforcement Learning

Artem Molchanov[1], Karol Hausman[1], Stan Birchfield[2], and Gaurav Sukhatme[1]

Abstract— **Learning a policy capable of moving an agent between any two states in the environment is important for many robotics problems involving navigation and manipulation. Applying reinforcement learning in these scenarios can be challenging, especially in the presence of sparse rewards. Common approaches to tackling this problem include reward shaping, which requires domain-specific knowledge and might result in changing the objective of the agent.**

We present a region-growing method for enabling an agent to learn transitions in an environment between any pair of initial and goal states. Our algorithm first learns how to move between nearby states and then increases the difficulty of the start-goal transitions as the agent's performance improves. This approach creates an effective curriculum for learning the objective behavior of reaching any goal from any initial state. Rather than use reward shaping, which requires domain-specific knowledge and often causes the objective of the agent to change, we use sparse rewards, which are a more natural but much more challenging approach. In addition, we describe a method to adaptively adjust expansion of the growing region that allows automatic adjustment of the key exploration hyperparameter to environments with different requirements. We evaluate our approach on a set of simulated navigation and manipulation tasks, where we demonstrate that our algorithm can efficiently learn a policy in the presence of sparse rewards.

I. INTRODUCTION

In recent years, deep reinforcement learning (Deep RL) has enjoyed success in many different applications, including playing Atari games [1], controlling a humanoid robot to perform various manipulation tasks [2], [3] and beating the world champion in Go [4]. The success and wide range of use cases of RL algorithms is partly due to the very general description of the problem that RL aims to solve, i.e., to learn autonomous behaviors given a high-level specification of a task by interacting with the environment. Such high-level specification is provided by a reward function, which must be sufficiently descriptive as well as easy to optimize for an RL algorithm to learn efficiently. These requirements make the design of the reward function challenging in practice, creating a bottleneck for an even wider set of applications for RL algorithms.

The problem of designing a reward function has been approached in various ways. These include: i) learning the reward function from human demonstrations in the field of inverse reinforcement learning (IRL) [5], [6], ii) initializing the reinforcement learning process with demonstrations in imitation learning [2], [7], and iii) creating reward shaping functions that aim to guide the RL process to high-reward regions [3], [8]. Although all of these methods have shown promising solutions to the problem of reward function design, they present other significant challenges such as the requirement of domain expertise or access to demonstrations.

Ideally, one would like to learn from a simple sparse binary reward that indicates completion of the task. Such a reward signal is natural for many goal-oriented tasks. It allows significant reduction of engineering effort, and in some cases can be used to learn complex skills from human feedback, where design of the reward function is challenging [9]. However, such a reward function creates significant difficulties for learning, because it is unlikely for an agent to generate the exact sequence of actions leading to solving the task by relying on random exploration [10].

Recent efforts focus on learning from such sparse reward signals by constructing a curriculum from a continuous set of tasks [11], [12]. These methods exploit the simple intuition that tasks initialized closer to the goal should be easier to solve. Proximity to the goal is defined either explicitly [11] or through the number of random actions needed to reach the state from the goal [12]. Nevertheless, these methods have a common limitation: they are designed for either single-start or single-goal scenarios. In this paper, we address the situation in which the task contains both a continuous set of goals and a continuous set of initial conditions, thus broadening the applicability of our algorithm to a wider range of problems. In addition, we introduce a method to adaptively adjust expansion of the growing region, eliminating manual tuning of a key exploration hyperparameter whose optimal value varies across different environments.

II. RELATED WORK

Intrinsic motivation. Learning from sparse rewards is a long-standing goal in RL. The most established way of coping with such scenarios has been reward shaping [3], which requires extensive engineering and domain specific knowledge. To address this problem, various researchers proposed curiosity and intrinsic motivation [13], [14] as a more general way of guiding learning in the absence of the task-specific reward. Intrinsic motivation is typically introduced in the form of auxiliary rewards or loss components incentivizing exploration, that are not connected to the main objective. Such incentives could be based on counting visited states and/or maintaining a state-visitation density model [15], [16], [17], prediction error [18], prediction error-improvement of the learned model [19], predictive model

[1] The authors are with the Department of Computer Science, University of Southern California, Los Angeles, CA {molchano, hausman, gaurav}@usc.edu

[2] Stan Birchfield is with NVIDIA, Redmond, WA sbirchfield@nvidia.com

uncertainty [20], neuro-correlation [21] or learning auxiliary tasks [22]. Despite a wide variety of approaches, many curiosity-inspired methods are prone to creating additional local minima in the learned objective function [23].

Curriculum learning. Another approach to learning in the presence of sparse rewards is to construct a *curriculum* of the task instances to ease the learning process. In this case, the agent initially learns from easy scenarios, where the chance of acquiring positive reward is relatively high, and the difficulty of the presented tasks is gradually increased until the final task is learned. The main advantage of such an approach is that the agent learns on the final objective directly, and thus avoids the problems of curiosity-driven methods. Traditionally, curriculum design has been explored from the perspective of manually engineered schedules in both supervised tasks [24], [25] and reinforcement learning scenarios [26], [27]. More recently, there have also been multiple approaches for automated curriculum generation for RL. In [28], the authors create curriculum in the form of an acyclic graph based on a *transfer potential* metric, [29] explore task sampling based on their current performance, and [30] utilize task performance improvement as a basis for task sampling. All of these approaches, as opposed to our method, are designed to perform well in a discrete set of tasks with dense rewards.

A few recent approaches leverage different types of prior knowledge about the system to construct an efficient curriculum for continuous sets of tasks. For example, [31] uses accuracy as the main parameter for constructing curriculum over possible transitions. Although being general, the notion of accuracy translates into a form of a known distance metric associated with the observation space explicitly used by the algorithm for curriculum learning. Another example is the work of [32] which leverages physical priors in the form of approximate system dynamics models to expand the initial state distribution.

Another related approach by [33] is based on the idea of self play between two agents. The first agent plays the role of a teacher that sets the tasks for the second agent, who plays the role of a student trying to repeat the teacher's actions or reverse the environment to its original state. As mentioned by the authors and confirmed in [12], the asymmetric structure of this method often leads to a biased exploration resulting in the teacher and the student becoming stuck in a small subspace of the task. Our method avoids such situations by using random exploration to expand the set of goals and the initial conditions to the appropriate level of difficulty.

Another work related to our approach is that of [11] that considers the problem of generating multiple goals of the appropriate level of difficulty using generative adversarial networks (GANs) [34]. Their approach is designed to learn a goal distribution and thus, in its straightforward form, cannot learn to generalize to multiple initial conditions. In addition, since their approach contains a learned generative model, it tends to struggle when the dimensionality of the task representation is large and the number of examples is limited, which is often the case for robotics. We address this problem by generating tasks through interaction with the environment.

The approach most related to ours is the work of [12]. We exploit similar core principles and assumptions, i.e., we utilize Brownian motion for growing the current task region and generate curriculum through reverse exploration of new tasks. We extend this approach to multi-goal and multi-start scenarios with infinitely many start-goal pairs, and present results in environments with sparse rewards. In addition, we address the question of controlling expansion of the growing region. Our algorithm adaptively changes the key exploration hyperparameter for environments with significantly different optimal settings. These contributions lead to improved resampling efficiency and eliminate the need of expensive hyperparameter tuning.

III. Background

We consider a reinforcement learning problem where an agent is represented by a global policy that aims to reach any goal in an environment. This section introduces a formal definition of the problem and our framework.

Markov decision process. We consider a discrete-time, finite-horizon Markov decision process (MDP) defined by a tuple $M = (\mathcal{S}, \mathcal{G}, \mathcal{A}, \mathcal{P}, r, \rho_0, T)$, in which $\mathcal{S}$ is the agent's state set, $\mathcal{A}$ is the action set, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^n$ is the transition probability distribution, $r : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to \mathbb{R}$ is a bounded reward function dependent on the goal state, where $\mathcal{G}$ represents the goal set; $\rho_0 : \mathcal{S} \to \mathbb{R}^n$ is the initial state distribution, and $T \in \mathbb{N}$ is the time horizon. Our aim is to learn a stochastic policy $\pi_\theta : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to [0, 1]$ parameterized by θ. In order to communicate the goal to the agent, our formulation requires the policy to be conditioned on the goal $g \in \mathcal{G}$ specified by the environment, i.e., $\pi_\theta = \pi(a_t|s_t, g)$. The objective is to maximize the expected return, $\eta_{\rho_0}(\pi_\theta) = \mathbb{E}_{s_0 \sim \rho_0, g \sim \rho_g} R(\pi_\theta, s_0, g)$ with the expected reward starting at s_0 being $R(\pi_\theta, s_0, g) := \mathbb{E}_{\tau|s_0}[\sum_{t=0}^{T} r(s_t, a_t, g)]$, where $\tau = (s_0, a_0, \ldots)$ denotes the trajectory generated by executing actions $a_t \sim \pi_\theta(a_t|s_t, g)$ sampled from the policy under environment dynamics $s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$.

Goal-dependent sparse reward function. We consider the problem of reaching any goal state $g \sim \mathcal{U}(\mathcal{G})$ in the environment from any initial state $\rho_0 \sim \mathcal{U}(\mathcal{S}^0)$, where $\mathcal{U}$ denotes a uniform distribution. For this purpose, we define a sparse binary reward function dependent on the goal:

$$r(s_t, a_t, g) = \mathbb{1}\{s_t \in S^g\}\,, \qquad (1)$$

where $S^g \subset \mathcal{S}$ is a set of states corresponding to the goal g. We note that although the binary reward function in Eq. (1) is typically defined through some distance metric ϵ, our learning algorithm does not explicitly utilize this metric.

Our method makes several assumptions that we state below:

Assumption 1. *The agent can be initialized at an arbitrary state* $s_0 \in \mathcal{S}$. This assumption is a common requirement for many algorithms [12], [35] in the RL setting, especially those that exploit uniform initialization to generalize to multiple initial states.

Assumption 2. *At least one initial state is provided to the algorithm, which we call a seed state.*

Assumption 3. *For every state $s \in \mathcal{S}$, there exists a function $g = f_g(s)$ that maps any state in the environment to the corresponding goal representation.* This assumption is required since, in our algorithm, states encountered by the agent should be converted to the corresponding goal representations.

Assumption 4. *For any pair of states $s_1, s_2 \in \mathcal{S}$ there exists a trajectory that moves the agent from s_1 to s_2.* In other words, the agent can reach any state from any other state.

Although we explicitly introduced Assumption 4, it does not prevent our algorithm from being applied to a wider set of tasks where some states might not be mutually reachable. For example, if isolated or irreversible pairs of states exist, the algorithm nevertheless can be applied to all the reachable states, which depend on the initial state provided.

IV. Approach

The main difficulty of training an RL agent in a sparse reward setting arises from the fact that it is unlikely for the agent to accomplish the task using random exploration if the initial state is far from the goal state. In this work, we take advantage of the intuition also exploited by [12] that the agent has a higher chance of success if the goal is located in close proximity to the initial state. In particular, initializing the learning process by generating goal states that are close to the initial states should enable the initial learning stages to progress much faster. Since it can be highly nontrivial to engineer a correct distance metric directly in the observation space, we define the proximity of points by the number of actions it takes to reach one point from another.

Taking this into consideration, we propose the idea of gradually-growing *reachability regions* for generating a curriculum in a multi-goal setting. Our algorithm consists of two agents: a sampler and a learner. The sampler uses short chains of random actions to arrive at a state that is then added to the currently-explored set of states, which we refer to as the reachability region. This region is defined as the area where the learner is currently mastering transitions between all pairs of points. As learning progresses, the sampler removes already-learned states from the reachability region and adds new points that have not yet been explored. This generates a natural curriculum for learning a global reaching policy, i.e., a policy capable of moving the agent between any two states in the environment. Thus, the goal of the sampler is to expand the reachability region and the goal of the learner is to master transitions between states within the reachability region. In the following, we first discuss the sampler and then the learner.

A. Filtering states

In the first part of the sampling algorithm, we focus on a criterion that indicates whether a particular set of states has been mastered. In order to select the mastered states, we retain statistics of rewards received by the agent on every state within the current reachability region. We choose to follow a simple approach, in which we only retain statistics of the points in the role of starting states, as opposed to retaining statistics on start-goal pairs. The algorithm uses thresholds R_{min} and R_{max} to reject overly hard or overly easy states, respectively. We refer to the set of all states in the current reachability region as s_r. Our algorithm keeps a history of rewards in a vector r and associates them with start states. If the average reward for a state in r does not exceed the R_{min} and R_{max} thresholds, we use the state for further resampling. This behavior is implemented in a helper function `FilterStates` that takes s_r, r, R_{min}, and R_{max} as input and returns the retained set of states as s_r.

B. Adaptive state resampling

As previously mentioned, we define the proximity of the points through the action space, i.e., points are close to each other if they are reachable via short random trajectories. We use Brownian motion to sample new states to grow the region of learned state-goal pairs.

A major challenge of this approach is the selection of the variance for exploration. Poorly selected variance can result in either a spread out set of points that are hard to learn from, or a set of points that are too easily mastered—both of which result in slow learning progress of the RL algorithm. We adjust the sampling variance σ^2 dynamically using a method that is inspired by the integral part of a PID controller. Our approach adjusts the variance such that the average reward in the current iteration (r_{avg}) is close to a user-provided target reward (R_{pref}). In particular, every time before resampling, we update the sampling standard deviation (σ) according to the following procedure:

$$\begin{aligned} \delta^\sigma &\leftarrow \text{Clip}(k_\sigma \cdot (r_{avg} - R_{pref}),\ -\delta^\sigma_{max},\ \delta^\sigma_{max}) \\ \sigma\ &\leftarrow \text{Clip}(\sigma + \delta^\sigma,\ \sigma_{min},\ \sigma_{max}) \end{aligned} \qquad (2)$$

where $\text{Clip}(x, \alpha, \beta) \triangleq \min(\max(x, \alpha), \beta)$, k_σ is the control coefficient, δ^σ_{max} is the maximum change of σ, and $\sigma_{min/max}$ are the limits. Thus, if the success ratio systematically exceeds the preferred value, our method increases the variance, promoting faster exploration and vice versa.

We encode Eq. (2) in the helper function `UpdateStd` that takes σ and r_{avg} as inputs and returns the new σ value. Resampling a set of new states is implemented in the helper function `ResampleS` (see Algorithm 1) that takes the current set of states s_r, the set of old mastered states s_{old}, and the variance σ^2 as inputs and returns the new set of states. Resampling is carried out in two stages. First, we create an oversampled set of states by performing Brownian-motion rollouts, which we refer to as sampling rollouts (lines 6 – 12). Random actions are generated by the sampler agent using $\mathcal{N}(\mathit{0}, \sigma^2 \cdot I)$ (where I is an identity matrix, $\mathit{0}$ is a zero vector) and collect the states visited by the agent. Each of these rollouts is initialized at one of the states from the growing oversampled set. This set is initialized with the states retained in s_r after filtering. At the second stage, we sample N_{new} states uniformly from the oversampled set and add them to the states sampled uniformly from s_{old} to form the new current set of states (line 13, where $\overset{N}{\sim}\mathcal{U}(s)$ denotes "sampling N times from a uniform distribution"). Lines 3–5 account for the scenario, where the algorithm rejects all samples at the previous iteration.

C. Policy training

Algorithm 1: ResampleS

```
   Input  : $s_r$, $s_{old}$, $\sigma^2$
   Output: $s_r$: states (growing region), $r$: rewards
1  $n_{old} \leftarrow 0$
2  # When failed to learn in previous iteration
3  if $len(s) = 0$ then
4      $s_r \leftarrow s_{old}$.getLastN($N_{old}$) , $n_{old} \leftarrow N_{old}$
5  end
6  while $len(s_r) < N_s \cdot (N_{old} + N_{new})$ do
7      $s_0 \sim \mathcal{U}(s)$, $a_0 \leftarrow 0$
8      for $t \leftarrow 1$ to $T$ do
9          $a_t \leftarrow a_{t-1} + \epsilon : \epsilon \sim \mathcal{N}(0, \sigma^2 \cdot I)$
10         $s_r$.append($s_t$) : $s_t \sim \mathcal{P}(s_t|s_{t-1}, a_t)$
11     end
12 end
13 $s_r$.append($\overset{N_{new}}{\sim}\mathcal{U}(s_r)$), $s_r$.append($\overset{N_{old}-n_{old}}{\sim}\mathcal{U}(s_{old})$)
14 $r \leftarrow [\,]$
```

Algorithm 2: Policy Training

```
   Input  : $s_{seed}$: seed state, $N$: iterations, $K$: sampling
            period, $\pi_1$: initial policy, $\sigma$: initial sampling
            standard deviation, $N_{new}$: number of new
            states in $s_r$, $N_{old}$: number of old states in $s_r$,
            $N_s$: state oversampling scale, $T$: rollout length
   Output: $\pi_{N+1}$: policy
1  $s_{old}, s_r \leftarrow [s_{seed}], [s_{seed}]$
2  $s_r$, $r \leftarrow$ ResampleS($s_r$, $s_{old}$, $\sigma^2$)
3  for $i \leftarrow 1$ to $N$ do
4      if $i$ mod $K = 0$ then
5          # Every $K$'th iteration
6          $\sigma \leftarrow$ UpdateStd($\sigma$, $r_{avg}$) # See Eq. (2)
7          # See Sec. IV-A
8          $s_r \leftarrow$ FilterStates($s_r$, $r$, $R_{min}$, $R_{max}$)
9          $s_{old}$.append($s_r$)
10         # See Sec. IV-B
11         $s_r$, $r \leftarrow$ ResampleS($s_r$, $s_{old}$, $\sigma^2$)
12     end
13     $s_{train}$, $g_{train}$, $r$, $r_{avg} \leftarrow$ Rollouts($\pi_i$, $s_r$, $s_{old}$)
14     $\pi_{i+1} \leftarrow$ UpdatePolicy($\pi_i$, $s_{train}$, $g_{train}$, $r$)
15 end
```

Algorithm 2 describes the policy training procedure including both the sampler and the learner agents. The sampler agent updates the reachability region (lines 5–12), while the learner follows its own learning strategy (lines 13–14).

Our method starts by initializing the current set of states s_r, the corresponding vector of history of rewards r and the pool of the previously learned states s_{old} (line 1).

The sampler uses a fixed update period K (line 4) to adjust the variance according to Eq. (2) (line 6) and proceeds to the filtering stage to find good states from which to propagate (line 8). Once the filtering is finished, the sampler resamples a new set of states using Brownian motion (line 11).

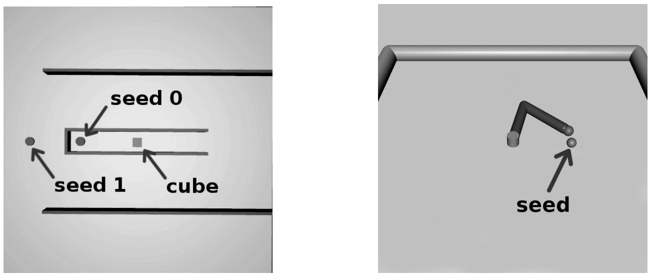


Fig. 1. Environments with seed states used in our experiments. Left: Maze environment. The square represents the cube that the agent has to push to a goal state. Black lines represent the walls of the maze. Right: SparseReacher environment. The two-link manipulator has to touch the goal marker.

The learner performs policy rollouts in every iteration (line 13) using the helper function `Rollouts`. This function follows a special start-goal pair sampling strategy. Start states of the rollouts are sampled uniformly from the current state set s_r, whereas the goals are sampled from either s_r (with probability P_{new}) or s_{old} (with probability $1 - P_{new}$). Once the batch of samples used for the user-chosen RL algorithm is accumulated, the policy is updated (line 14).

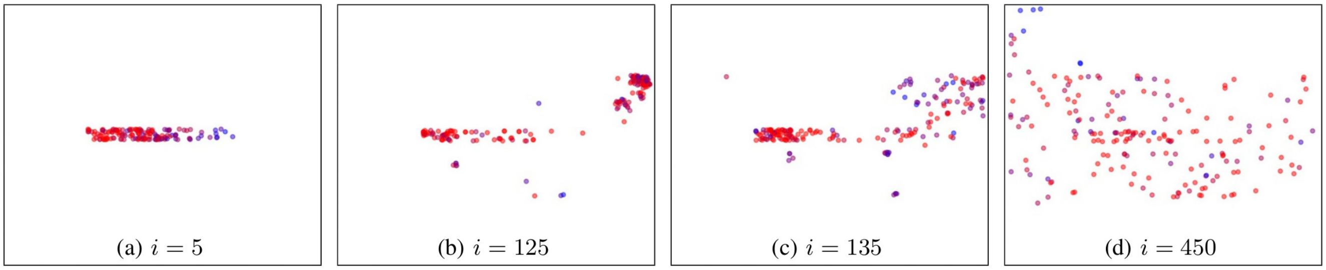
Our approach is agnostic to the choice of agent optimization method—we only require that this method provides an `UpdatePolicy` function. In our experiments, we use TRPO [36] as one of the most robust RL algorithms with an implementation available online.

V. Experiments

We apply our approach to two representative environments. We show empirically that this technique successfully trains agents in the multi-goal scenarios. Furthermore, we demonstrate that our dynamic variance selection is less sensitive to hyperparameters than other alternatives. In all of our experiments, we use the following parameters across all environments: $R_{max} = 0.9$, $R_{min} = 0.3$, $K = 5$, $N_s = 5$, $N_{new} = 135$, $N_{old} = 65$, $P_{new} = 0.6$, $R_{pref} = 0.7$, $k_\sigma = 2.0$, $\delta^{\sigma}_{max} = 0.5$, $\sigma_{min} = 0.1$, $\sigma_{max} = 1.0$. Our algorithm shows very mild sensitivity to the hyperparameters mentioned above. The hyperparameter values were selected either empirically or based on recommendations for similar hyperparameters from [12].

A. Environments

The *SparseReacher* is an environment with a two-link manipulator based on `Reacher-v0` from OpenAI Gym [37]. We use it in a sparse reward setting: the agent receives a positive binary reward only when it touches the goal marker. This corresponds to the situation where the robot's end effector is not further than $2\,\mathrm{cm}$ from the center of the goal marker. In addition, the Cartesian velocities of the robot must be lower than $0.2\,\mathrm{m/s}$. The episode ends when the positive reward is acquired. As we observed in our experiments, such sparse reward makes this environment significantly more challenging, especially when the goal is to learn a policy that can reach any point in the robot's workspace.



(a) $i = 5$ (b) $i = 125$ (c) $i = 135$ (d) $i = 450$

Fig. 2. Illustration of state propagation for the maze multi-goal environment. Circles represent the current states in the reachability region. Images are ordered from left to right in the order of learning progress. The middle two plots show the phenomenon of state clustering. Colors encode average reward associated with the states, where red refers to high reward and blue to low reward.

The goal in the *Maze* environment is to bring a cube of size h_{cube} to a goal location. The agent receives a reward only if the center of the cube lies still within an ϵ-radius of the goal location. The episode ends as soon as the positive reward is acquired. We define a variable time step in the environment that is dependent on the time it takes for the cube to settle after a force is applied. The table is constrained by the size h_{table} and surrounded by walls, such that the cube cannot fall off the table. This environment has continuous action space that consists of two components of the force F_x, F_y applied to the center of the cube, parallel to the table plane. Observations contain a 7-dimensional cube pose where the rotation is encoded as quaternion. We define a goal representation as a simple 2d position on the table.

This environment is challenging due to several aspects. First, the search space increases with h_{table}^2, thus, the probability to encounter the target by chance is very small. Second, the cube has complex dynamics compared to a simple point mass: it can be pushed or rolled depending on the direction and amount of force applied, and it exhibits a complex behavior when it comes into a contact with the wall. Third, the cube must stop at the precise location of the goal.

Both environments are shown in Fig. 1. For each environment, we select a single *seed* state to expand the growing region. For the Maze environment, we explicitly pick the most challenging scenario of the seed state located at the end of the central corridor (seed 0) since the policy has to learn how to precisely navigate inside of the narrow corridor entrance. Both environments can be naturally used in both single- and multi-goal settings. In every training scenario, we add a very small negative reward for every time step to promote shorter episodes, in addition to the sparse reward.

B. *Reachability regions*

Fig. 2 demonstrates the region expansion during learning in the maze environment. In particular, it shows an interesting phenomenon associated with variance adaptation that we refer to as region clustering. During expansion, if the new set of points was selected too aggressively, our algorithm responds by decreasing the variance of the region expansion. Since this event by definition happens due to a poor performance (see Eq. (2)), there will be very few available states to sample from. Thus, the algorithm forms a cluster of newly resampled states located around a few states that passed through the filtering stage (see Fig. 2(b)). Later, as the learner agent improves, those clusters grow and connect, forming a single region which is illustrated in Fig. 2(c). Such behavior helps the learner to create new growing regions in isolated areas.

C. *SparseReacher*

Our results for the multi-goal version of the SparseReacher environment are shown in Fig. 3. We execute the learning process several times and provide the average reward for each iteration over ten executions. We test our algorithm with $\sigma = 0.1$ and $\sigma = 0.5$ and with our adaptive variance. We also provide results for the case that does not use a reachability region, but instead samples start and goal states uniformly over the environments (i.e. without curriculum). In addition, we provide comparison to the multi-goal version of the selfplay approach [33].

The environment is conservative and requires small exploration variances; we found that a constant $\sigma = 0.1$ performs much better than $\sigma > 0.5$. Our adaptive variance selection achieves a slightly higher average reward than the best hand-tuned constant variance. The uniform state sampling performs as well as our reachability approach when a bad constant variance is applied. The selfplay approach has the highest learning curve at the beginning, but it fails to explore and converges to a suboptimal equilibrium of the the student and the teacher. The tendency to being stuck in a suboptimal minima is also mentioned in other sources [12], [33].

D. *Maze*

Our results for the multi-goal version of the Maze environments are shown in Fig. 3. As before, we provide the average reward for each iteration over ten executions.

This environment requires more exploration than the SparseReacher environment; we found that when using a constant value of $\sigma = 1.0$ the agent performs best, while $\sigma < 0.25$ results in a very poor learning performance. The performance of our algorithm in this environment also depends on the seed state. For comparison we provided the hardest (seed 0) and the easiest (seed 1) case comparison. The reward of our adaptive variance selection is comparable to the best hand-tuned constant variance.

Uniform state-goal sampling performed surprisingly well, but our approach clearly indicates the benefits of generating a curriculum for learning and outperforms uniform sampling even in the scenario with the hardest seed.

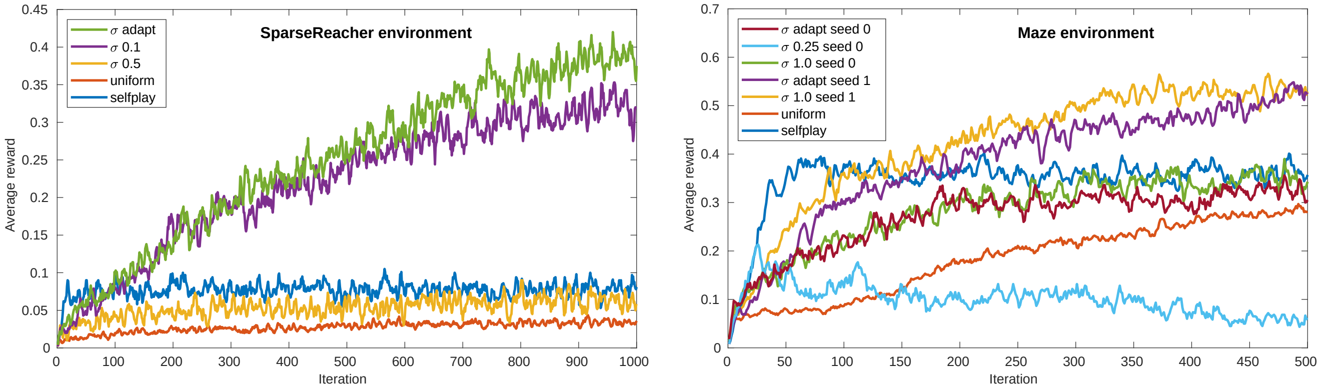


Fig. 3. Reward for different algorithm variants for the multi-goal case. The data is averaged over 10 executions. "Uniform" refers to uniform random sampling of the start and goal states with no reachability region (no curriculum). "selfplay" refers to the asymmetric selfplay algorithm presented by [33]. "σ x" uses the reachability regions for the sampler agents, but assigns a constant $\sigma = x$ for action selection. "σ adapt" is our full algorithm using the reachability regions and adaptive σ.

The selfplay approach shows a very steep learning curve at the beginning, but, similar to the reacher environment, it fails to fully explore the state space and saturates approximately at the level of the seed 0 scenario of our algorithm.

E. Hyperparameter adaptation

The environments that we selected are representative in the spectrum of requirements for growing region expansion. The multi-goal version of the SparseReacher environment is more conservative and requires small exploration variances, whereas the Maze environment benefits from aggressive exploration, and hence high variances perform well. For example, Fig. 3 (left) shows that, under constant variance, the learner completely fails to improve when the variance is set to a high value. On the other hand, Fig. 3 (right) shows the opposite for the Maze, where the optimal variance value is close to the maximum value. Our adaptive variance approach performs similar to the optimal constant variance. Given that we have the same set of exploration hyperparameters for both environments, our approach eliminates the need to tune the key hyperparameter of the region growing curriculum learning method.

Fig. 4 shows the sampling variance evolution over training. Initially, our algorithm picks the largest and the smallest variance values for the Maze and the SparseReacher environments, respectively. In the case of SparseReacher, it keeps a low variance at the beginning, since random initialization of the policy weights results in actions of large magnitude. As the agent keeps learning, the exploration is gradually relaxed. Our algorithm regulates the variance in such a way that allows the learner to maintain the proper exploration pace, resulting in steep learning curves. We also find this idea connected to the approach proposed by [38] in the context of adversarial learning. In our scenario, since there is no loss for the sampler, we apply the equilibrium principle through balancing the success ratio for the learner.

We also evaluated single-goal variations, where the seed state represents the only goal in each environment. In this scenario, variance adaptation showed similar benefits. On average the single-goal SparseReacher was learned 20–50% faster with variance adaptation than with manual tuning of a constant variance. For the Maze environment our algorithm is able to match the performance of the version with the constant sampler variance.

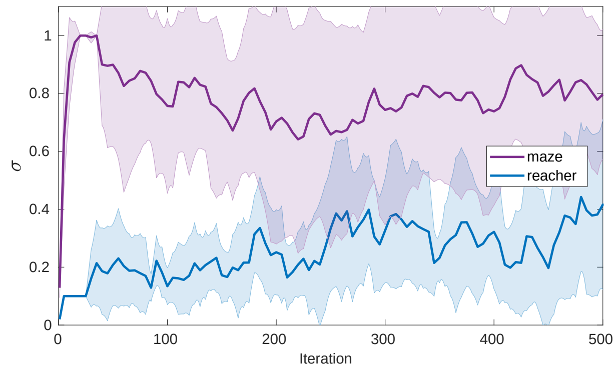


Fig. 4. σ adaptation for different environments in the multi-goal scenario. The lines show the average and the shaded region the standard deviation over 10 executions.

VI. Conclusions and Future work

In this work, we propose a novel algorithm for learning a global policy capable of moving an agent in environments with any pair of start-goal transitions. Our algorithm is based on the idea of region growing and it has several attractive properties, such as: i) our approach is agnostic to the choice of the agent optimization method, ii) it does not require any strong priors about the system, iii) it is capable of constructing effective curriculum in environments with continuous states and actions. One of our key contributions is employing automatic adjustment of the region expansion that results in appropriate pace of learning without extensive hyperparameter tuning.

Our investigation revealed a few interesting directions for future work. First, the algorithm could substantially benefit from parallel learning of a reversing policy, allowing it to return to safe states within the current growing region. Second, the current version of our algorithm is sensitive to the choice of the seed state. We plan to address this problem by utilizing a hierarchical approach and learn a set of local policies for different state regions.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," in *NIPS Deep Learning Workshop*, 2013.

[2] Y. Chebotar, M. Kalakrishnan, A. Yahya, A. Li, S. Schaal, and S. Levine, "Path integral guided policy search." in *ICRA*, 2017.

[3] Y. Chebotar, K. Hausman, M. Zhang, G. S. Sukhatme, S. Schaal, and S. Levine, "Combining model-based and model-free updates for trajectory-centric reinforcement learning," in *ICML*, 2017.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[5] S. Levine, Z. Popovic, and V. Koltun, "Nonlinear inverse reinforcement learning with Gaussian processes," in *Advances in Neural Information Processing Systems (NIPS)*, 2011, pp. 19–27.

[6] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the Twenty-first International Conference on Machine Learning (ICML)*, 2004.

[7] M. Kalakrishnan, L. Righetti, P. Pastor, and S. Schaal, "Learning force control policies for compliant robotic manipulation," in *Proceedings of the 29th International Conference on Machine Learning (ICML)*, 2012.

[8] I. Popov, N. Heess, T. P. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. A. Riedmiller, "Data-efficient deep reinforcement learning for dexterous manipulation." *arXiv 1704.03073*, 2017.

[9] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *NIPS*, 2017.

[10] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2016, pp. 1329–1338.

[11] C. Florensa, D. Held, X. Geng, and P. Abbeel, "Automatic goal generation for reinforcement learning agents," in *ICML*, 2018.

[12] C. Florensa, D. Held, M. Wulfmeier, and P. Abbeel, "Reverse curriculum generation for reinforcement learning," in *CoRL*, 2017.

[13] J. Schmidhuber, "Formal theory of creativity, fun, and intrinsic motivation (1990-2010)," *IEEE Trans. Autonomous Mental Development*, vol. 2, no. 3, pp. 230–247, 2010.

[14] P. Oudeyer, F. Kaplan, and V. V. Hafner, "Intrinsic motivation systems for autonomous mental development," *IEEE Trans. Evolutionary Computation*, vol. 11, no. 2, pp. 265–286, 2007.

[15] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 1471–1479.

[16] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos, "Count-based exploration with neural density models," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2017, pp. 2721–2730.

[17] J. Martin, S. N. Sasikumar, T. Everitt, and M. Hutter, "Count-based exploration in feature space for reinforcement learning," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2017, pp. 2471–2478.

[18] B. C. Stadie, S. Levine, and P. Abbeel, "Incentivizing exploration in reinforcement learning with deep predictive models," *arXiv 1507.00814*, 2015.

[19] M. Lopes, T. Lang, M. Toussaint, and P. Oudeyer, "Exploration in model-based reinforcement learning by empirically estimating learning progress," in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 206–214.

[20] R. Houthooft, X. Chen, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel, "VIME: Variational information maximizing exploration," in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 1109–1117.

[21] J. Schossau, C. Adami, and A. Hintze, "Information-theoretic neurocorrelates boost evolution of cognitive systems," *Entropy*, vol. 18, no. 6, 2016.

[22] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," in *ICLR*, 2017.

[23] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in AI safety," *arXiv 1606.06565*, 2016.

[24] W. Zaremba and I. Sutskever, "Learning to execute," *arXiv 1410.4615*, 2014.

[25] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 1171–1179.

[26] Y. Wu and Y. Tian, "Training agent for first-person shooter game with actor-critic curriculum learning," in *International Conference on Learning Representations (ICLR)*, 2017.

[27] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," *arXiv 1707.02286*, 2017.

[28] M. Svetlik, M. Leonetti, J. Sinapov, R. Shah, N. Walker, and P. Stone, "Automatic curriculum graph generation for reinforcement learning agents," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017, pp. 2590–2596.

[29] S. Sharma, A. Jha, P. Hegde, and B. Ravindran, "Learning to multitask by active sampling," *arXiv:1702.06053*, 2017.

[30] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman, "Teacher-student curriculum learning," *arXiv:1707.00183*, 2017.

[31] P. Fournier, O. Sigaud, M. Chetouani, and P.-Y. Oudeyer, "Accuracy-based curriculum learning in deep reinforcement learning," *arXiv 1806.09614*, 2018.

[32] B. Ivanovic, J. Harrison, A. Sharma, M. Chen, and M. Pavone, "BaRC: Backward reachability curriculum for robotic reinforcement learning," *arXiv 1806.06161*, 2018.

[33] S. Sukhbaatar, I. Kostrikov, A. Szlam, and R. Fergus, "Intrinsic motivation and automatic curricula via asymmetric self-play," in *ICLR*, 2017.

[34] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 2672–2680.

[35] M. J. Kearns, Y. Mansour, and A. Y. Ng, "A sparse sampling algorithm for near-optimal planning in large markov decision processes," *Machine Learning*, vol. 49, no. 2-3, pp. 193–208, 2002.

[36] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2015, pp. 1889–1897.

[37] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv 1606.01540*, 2016.

[38] D. Berthelot, T. Schumm, and L. Metz, "BEGAN: boundary equilibrium generative adversarial networks," *arXiv 1703.10717*, 2017.